ICT PROJECT 258109

Monitoring Control for Remote Software Maintenance

<table>
<thead>
<tr>
<th>Project number:</th>
<th>258109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Acronym:</td>
<td>FastFix</td>
</tr>
<tr>
<td>Project Title:</td>
<td>Monitoring Control for Remote Software Maintenance</td>
</tr>
<tr>
<td>Call identifier:</td>
<td>FP7-ICT-2009-5</td>
</tr>
<tr>
<td>Instrument:</td>
<td>STREP</td>
</tr>
<tr>
<td>Thematic Priority:</td>
<td>Internet of Services, Software, Event Processing</td>
</tr>
<tr>
<td>Start date of the project:</td>
<td>June 1, 2010</td>
</tr>
<tr>
<td>Duration:</td>
<td>30 months</td>
</tr>
</tbody>
</table>

D4.1: State-of-the-art of event correlation and event processing

Lead contractor: S2 GRUPO
Sergio Zamarripa López, María del Carmen Calle

Author(s): Villanueva, Emitzá Guzmán, Tobias Röhm, Benoît Gaudin, Newres Al Haider

Submission date: December 2010

Dissemination level: Public

Project co-founded by the European Commission under the Seventh Framework Programme

© S2, TUM, LERO, INESC ID, TXT, PRODEVELOP
Abstract: We examine the state of research and practice regarding event correlation and its application to software maintenance, documented and discussed in order to find out the best alternatives to meet FastFix requirements. Event Correlation will play an important role, since it must process user interaction and system context data to draw conclusions about the problem being faced by the application under monitoring. Current event correlation techniques will be described as well as its use in the field of software error detection and cause identification.

This document has been produced in the context of the FastFix Project. The FastFix project is part of the European Community’s Seventh Framework Program for research and development and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.
Contents

1 Introduction 7

2 Foundations 9
  2.1 Basic Concepts ......................................................... 9
  2.2 Event-driven Architecture ........................................... 11
  2.3 Complex Event Processing versus Event Stream Processing .... 12
  2.4 Pattern Recognition and Pattern matching ......................... 14
  2.5 Root Cause Analysis ................................................... 15
  2.6 Fault Prediction .......................................................... 16

3 Event Correlation Techniques 20
  3.1 Main Event Correlation Techniques .................................. 20
    3.1.1 Rule Based Event Correlation ................................... 20
    3.1.2 Case Based Event Correlation ................................... 21
    3.1.3 Finite State Machine Based Correlation ......................... 22
    3.1.4 Model Based Correlation .......................................... 23
    3.1.5 Probabilistic Event correlation .................................. 24
    3.1.6 Codebook Based Event Correlation ............................... 27
    3.1.7 Comparison of the Main Event Correlation Techniques ........ 28
  3.2 Semantic Event Correlation ............................................ 29
  3.3 Main Issues ............................................................... 33
  3.4 Learning ................................................................. 34
    3.4.1 Mining Association Rules to Find Patterns and Relationships between Events ............................................... 34
    3.4.2 Learning Sequential Patterns ..................................... 35
    3.4.3 Learning Patterns for Rule-based Correlation .................. 36
    3.4.4 Learning for Case-based Correlation .................................. 38
    3.4.5 Learning Patterns for Probability-based Correlation .......... 38
4 Existing Systems and Applications

4.1 Event Correlation Frameworks
4.1.1 Existing Open Source Event Correlation Software
4.1.2 Commercial Event Correlation Software
4.1.3 Summary

4.2 Event Correlation Applications
4.2.1 OSSEC
4.2.2 HP Event Correlation Services
4.2.3 OpenNMS
4.2.4 Open Source Security Information Management
4.2.5 Prelude-Intrusion Detection System

4.3 Related EU projects
4.3.1 ONTORULE
4.3.2 Other EU Projects Related with Event Correlation

5 Conclusion

Bibliography
## List of Figures

3.1 System G and Diagnoser $G_d$. ......................................................... 24  
3.2 An example of Bayesian Network for WiFi Connection Issues ............ 26

4.1 Borealis System ................................................................. 43
4.2 Borealis Node Components [1] ............................................... 43
4.3 Triton-Web Interface: Event Statistics ....................................... 48
4.4 Basic Rete Network ............................................................ 50
4.5 ILOG JRules Scenario Manager ............................................... 52
4.6 Oracle CEP Visualizer Dashboard ........................................... 54
4.7 Prelude Simplest Architecture ............................................... 64
4.8 PR(OWL) Rule Example ....................................................... 66
List of Tables

3.1  A possible conditional probability table for node “Page not Displayed” .... 27
3.2  Example of codebook correlation matrix ................................. 28
3.4  Comparison of existing techniques [55] ................................. 29

4.1  Comparison of existing event correlation frameworks (Part 1 of 2) .... 56
4.2  Comparison of existing event correlation frameworks (Part 2 of 2) .... 57
1 Introduction

In our days, information processing using computer systems has become essential in our life. Occurrences of changes of state, alerts, warnings, faults, network access violations and so on flow silently through our systems without being recognized. Nevertheless, they are a source of great power, since when they are aggregated together and their relationship is understood, they yield a wealth of useful information, allowing the inference of meaningful knowledge that can become the key for strategic decisions, like re-positioning organizations to take full advantage of business emerging opportunities.

Event correlation is the measure of the relationship between events in order to make sense of a large number of them, first discarding irrelevant events, merging duplicates of events and triggering actions with the information provided by the most relevant events. This task and the need of computing applied to events leads to the concept of event processing.

The primary goal of FastFix is to facilitate a more time and cost-efficient software maintenance. Within this project, event correlation will be in charge of determining how the gathered information is going to be used to draw conclusions about the kind of problems the monitored application is facing and what possible causes have led to the current situation of fault or performance deterioration.

Since event correlation has traditionally been used to process monitoring events in other areas than software maintenance, we survey the state of the art in order to gather the most feasible techniques for events associated to software errors, discussing between pros and cons of those techniques and the correlation engine strategy to process FastFix events. These events will be composite events, since failure will often have complex causes due not only to application errors, but also to problems in operating systems or execution environments or even to changes in user behaviour. Another relevant consideration is the need to take into account not only the information available in the events being correlated at a point of time, but also the previous event history, so special consideration will be paid on techniques like sliding time windows in order to correlate environment and application execution data. In some cases, related with fault prevention and pattern matching, FastFix will require real-time response to the occurrence of certain events. Therefore, special attention will be paid to the research of supporting technologies able to provide this real-time processing.
The use of Event Correlation in FastFix responds to the need of processing incoming events related to fault symptoms and effects in a quasi real-time, identifying current situations by means of a suitable configuration language (language choices will also be listed). The final goal is to identify known event patterns, as well as to filter, group and prioritize the events in order to infer what is the current situation of the system under monitorization.

Martin-Flatin [51] claimed that information systems should evolve into self-managed systems, detecting problems by themselves, working out the cause of each problem and taking corrective actions. In that sense, the current state of learning event correlation systems will be investigated, as well as other concepts like root cause analysis and fault prediction.

The present document aims at capturing the current state of research and practice regarding event correlation, discovering the main techniques and real implementations of these techniques, playing special attention to its application to software maintenance.

This state of the art document is structured into 5 chapters. The following Chapter 2 presents the foundations of the event correlation field defining some basic concepts that will be used along the road. In Chapter 3 we will introduce the main event correlation techniques in order to give a bird’s eye view of this discipline, as well as the point of view of software maintenance as target environment, while Chapter 4 focuses on real implementations of event correlation frameworks and some applications using these frameworks as final systems applying event correlation. Finally, in Chapter 5, we will draw some conclusions of this topic.
2 Foundations

In this section we provide the most important definitions of concepts related with the event correlation discipline. We also describe the main activities in event correlation, connecting them with software maintenance, in order to classify the main functionalities that event correlation can provide in the scope of FastFix.

2.1 Basic Concepts

The first definition that comes to our minds is the concept of event. An event is an occurrence within a particular domain; it is something that has happened or is contemplated as having happened in that domain[22]. Luckham[48] defines it as a record of an activity in the system, with three aspects: form, significance and relativity. First, about the form aspect, it refers to their attributes or data components, in other words, the form of an event is an object. About significance, it means the activity that the event is representing. Finally, Luckham[48] describes the relativity aspect as the way the event is related to other activities by time (a relationship that orders the events), causality (some events happen because other occurred) and aggregation (a complex event is an abstraction of a set of lower level events). Events have the same relationships to one another as the activities that each of them signify. In the context of computing, the term event is also used for the programming entity that represents the occurrence, very close to the form aspect described by Luckham. Focusing on software maintenance, events can be viewed as generalized log records produced by various sensors, and can be related to a change in the state of the operating system, the application or the user interaction.

Depending on the origin and nature of the event, we can categorize events in the following types[55]:

- **Primitive event or raw event**: An event generated by an event source, whose origin is outside the correlation engine.

- **Derived event**: An event generated by the correlation engine, for example, because a given event was not generated for a specified time (e.g. timeout event) or because a given set of primitive events occurred.
D4.1: State-of-the-art of event correlation and event processing

- **Composite event**: An event generated by the correlation engine containing other events (raw, derived or composite).

- **Compressed event**: An event representing multiple identical events, without containing the individual events it represents.

The next step to be completed is the definition of event correlation. As we have mentioned in the introduction, event correlation is needed to gain higher level knowledge from the information in the events. The term correlation becomes clearer, by inserting a hyphen at the right place [55]: we are looking for co-relation, in other words, for relations between different events, the ‘relativity’ that Luckham[48] mentioned in its definition. Another concept that must be clarified before formally defining “event correlation” is the concept of “event processing”. The definitions of event processing provided by Etzion [22] and event correlation, defined by Müller [55], are the following:

“Event processing is the discipline of computing that performs operations on events. Common event processing operations include reading, creating, transforming and deleting events.”

“Event correlation is a technique with the purpose of gaining higher level knowledge from the information in a set of events.”

The relationship between event processing and event correlation is explained in Softpanorama¹, an Open Source Software Educational Society, where introducing event correlation technologies, it claims that event correlation is one of the most important parts of event processing flow, critical to ensuring service quality and the ability to respond rapidly to exceptional situations.

Tiffany stated [76] that event correlation, in essence, tries to do exactly what its name suggests, that is, associating events with others in useful ways. The need of it can be perfectly understood if we think of a huge number of events, enough to overwhelm a engineer who cannot possibly treat each symptom separately. The purpose of event correlation is to attempt to pinpoint larger problems which could be causing many different symptoms to emerge.

About the higher level of knowledge that event correlation can provide, some examples are the identification of extraordinary situations, the prediction of future situations or the identification of the root cause of the problem, which are directly applicable to the environment of software maintenance. In order to perform these functionalities[55], event correlation can be decomposed into four steps:

- **Event filtering**: For the purposes of the event correlator, it consists of discarding events that become irrelevant to the occurrences to be observed.

¹http://www.softpanorama.org/Admin/Event_correlation/
D4.1: State-of-the-art of event correlation and event processing

- **Event aggregation**: It is based on merging duplicates of the same event.

- **Root cause analysis**: It is the task of analyzing dependencies between events, usually based on a model of the environment and dependency graphs, to detect whether some events can be explained by others.

- **Event masking**: It deals with ignoring events pertaining to systems that are downstream of a failed system.

In addition to these four steps, we should mention the action triggering which is the task in charge of including problem-solving capabilities, like corrective actions.

### 2.2 Event-driven Architecture

The major architectural component of an event-driven architecture, from a general point of view, contains a set of event producers, a set of event consumers and both are linked by some kind of event distribution and processing mechanism[22]. The event producers, also known as event sources[55], are entities that emit events. These producers can be systems, applications, business processes or sensors that are monitoring them. About the event consumers[22], also known as event sinks[55], they are entities that receive events. The idea of decoupling event producer and consumer is a significant difference between event-based programming and the request-response invocation pattern, and it allows event processing to be performed asynchronously to event arrival, so it is well suited to applications where events happen in an irregular manner. In addition to that, it supports one-to-many and many-to-one message exchanges, in addition to the one-to-one exchange found in the request-response pattern.

An event processing network is a collection of event processing components, producers, consumers and global states, connected by a collection of channels[22]. Event Processing is not monolithic, but is composed of several event processing components (EPC), which are software modules that process events, in charge of reading, creating, transforming and deleting events. These components are specified in an event processing language (EPL), and they can be rules-based, script-based and SQL extensions, among others.

The type of the language is, in most of cases, strongly subordinated to the technology of the correlation system and the correlation implemented by it.

The logical functions of an event processing component are[22]:

- **Filtering**: Selecting which of the input events participate in the processing.

- **Matching**: Finding patterns among events and creating sets of events that satisfy the same pattern.
Derivation: Using the output from the matching step to derive new events and setting their content.

The main types of event processing components are the following:

- **Filter**: It performs filtering only and has no matching or derivation steps, so it does not transform the input event.

- **Transform**: It performs the derivation function, and optionally also the filtering function.

- **Pattern Detection**: It performs a pattern matching function on one or more input streams. It emits one or more derived events if it detects an occurrence of the specified pattern in the input events.

- **Translate**: A stateless transform EPC that takes as an input a single event, and generates a single derived event which is a function of the input event, using a translation formula.

- **Aggregate**: A transform EPC that takes as input a collection of events and creates a single derived event by applying an aggregation function over the input events.

- **Split**: A transform EPC that takes as an input a single event and creates a collection of events, each of them can be a clone of the original event, or a projection of that event containing a subset of its attributes.

- **Compose**: A transform EPC that takes groups of events from different input terminals, looks for matches using some matching criterion and then creates derived events based on these matched events.

- **Enrich**: A translate EPC that takes a single input event, matches it against other existing event, and creates a derived event which includes the original event, with possible modified attributes, and an additional collection of attributes copied or calculated as a result of using the global state.

- **Project**: A subtype of the translate EPC that takes an input event, and creates a single derived event that contains a subset of the attributes of the input event.

### 2.3 Complex Event Processing versus Event Stream Processing

In the context of event correlation, two terms are usually mentioned as key concepts of this discipline: Complex Event Processing (CEP) and Event Stream Processing (ESP).
D4.1: State-of-the-art of event correlation and event processing

In ESP, a single event does not trigger reaction, since stream analysis is required. In that sense, it uses sliding windows (time based and size based) to perform this analysis. In CEP, events are processed as part of a complex analysis of events in a cloud of events. A cloud of events is really a bunch of streams of events, maybe multiplexed and maybe distributed. As we have mentioned in the event categories, a complex or composite event is an event that could only happen if lots of other events happened [48], so it can be understood as a higher level event, more meaningful than primitive events and which nature is artificial, it is not generated by an event source but by the correlation engine, since a given set of primitive events occurred. Complex events can be inferred because events are related in various ways: by cause, by timing or by membership among others. Correlation engines make use of relationships between events to answer complex questions, giving complex events as a result.

Ayllon et al. [9] claimed that ESP’s goal is processing and capturing of a big amount of event in a concrete time window (event stream). This stream is conceived as a sorted sequence of events of the same type. Instead, the aim of CEP is processing and capturing different type events in the cloud of events (whose event are not sorted at all). The cloud is the result of many events generated by all the activities going on at different places in an IT system 2. In that sense, ESP is a subset of CEP, since a cloud of events can contain many event streams and a stream is a special case of cloud.

An ESP/CEP architecture must be capable of processing clouds and streams of events originated by one or several event producers, storing and classifying events in a repository, keeping time of occurrence into account, providing mechanisms for complex pattern detection and generating automatic responses to the event consumers of the detected patterns.

Ayllon [9] also provides an example of ESP/CEP which takes place in the stock exchange:

• With ESP, we could be able to obtain the average price of market shares in a concrete event stream, in other words, in a time window.

• With CEP, we could go further and define complex patterns correlating information about an event of a company in the news with the growth or descent of the average price of its market shares.

Hence, CEP has to model not only the timing of events as they were created, but also their causal relationships or their independence.

Thinking about simplicity, ESP has some advantages derived from the fact or processing a stream of events in their order of arrival, since it allows the use of algorithms for

---

D4.1: State-of-the-art of event correlation and event processing

processing the data that use a small amount of memory because they don’t have to remember many events. Thus, these algorithms can be really fast. On the other side, CEP scenario characterizes by a lack of a proper order of the arrival of events, so the engine will be forced to remember lots of events before finding the set of events that have a complex relationship with a particular one, like causality. In that case, this process takes considerably more memory and more time.

Forgetting about simplicity, and focusing on applications of both approaches, CEP can deal with a richer set of problems like business process management, extracting information from clouds of events created in enterprise IT and business systems. It also remarks emphasis on patterns of events, trying to abstract and simplify information in the patterns and detecting occurrences along the cloud. Otherwise, ESP can concentrate on high-speed querying of data in streams of events, so historically it was first applied to stock-market feeds in financial systems. At present, ESP tools are evolving through the algorithmic trading area. They fit were customers can’t predict in advance where their processing needs will evolve.

From the point of view of software maintenance, ESP approach will be useful to monitor database transactions, while CEP would be used for example in any complex processing concerning root cause analysis of a software fault.

2.4 Pattern Recognition and Pattern matching

When the objective is the anticipation of specific events, like failures, unavailability of systems, performance degradation, the solution is pattern recognition. This discipline aims to classify data patterns based on previous knowledge or statistical information extracted from the patterns. The difference with other related discipline like pattern matching is that the matching refers to a rigidly specified pattern and it usually must be exactly matched.

But before going deeper into these disciplines, we should define the concept of pattern. Etzion [22] claims a pattern is a function that takes a collection of input event instances and produces a matching set that consists of zero or more of those input events.

The concept of pattern matching is defined by Agrawal [3] as a processing paradigm where continuously arriving events are matched against complex patterns, and the events used to match each pattern are transformed into new events for output.

One of the first noticeable things of pattern matching is that languages used in this discipline are significantly richer than languages for regular expression matching [3]. They usually contain constructs for expressing sequencing, negation, complex predicates and strategies for selecting relevant events from an input stream.

As we described in section 2.2, one of the main types of event processing components
is the pattern detection, which examines the incoming event stream looking for a combination of events that matches a pre-defined pattern. This pattern matching process has three main steps [22]. First, filtering, which is about the selection of the relevant events. Second, matching, which is a selection of subsets of these events. And finally, derivation, when the output from the matching step is taken and it is used to derive one or several new events, which is defined as the pattern matching set.

Event correlation systems, in most of cases, must provide a way to support sophisticated pattern matching on real time event streams, especially when tracking time sensitive applications, where a second makes a difference, like real-time intrusion detection[47].

Real time pattern matching is provided depending on the technology selected for event correlation, so it will be included as one of the main criteria in order to evaluate the existing event correlation technologies in this document.

2.5 Root Cause Analysis

A root cause is the underlying original fault that has led to a particular incident[56]. Root cause analysis tries to map an incident to its underlying fault. In other papers, root cause analysis has also been commonly referred to as fault localization, fault diagnosis, and fault identification. This section is strongly connected to research performed within the Fault Replication workpackage of FastFix (deliverable D5.1).

A causality or dependency graph is a directed graph, which models dependencies between the managed objects, represented by nodes [55]. It stores the cause and effect relations between different components of a system, often modeling the strength of the relationship with the use of probabilities. The use of dependency graphs for event correlation is frequently associated to the concept of root cause analysis. The goal is to find the likely root cause of some fault events generated in a time window.

Event correlation also uses the concept of event causality as the relation between two events, designating the fact that the occurrence of the first of them, caused the occurrence of the second. In the context of event correlation, this concept has practical importance, since it is possible to trace back the events that led to the execution of some action through causality relations. Etzion et al.[22] defined the following types of event causality:

- Predetermined causality: Take two raw events, where e2 always occurs as a result of the occurrence of e1. It can be assumed that if e1 has been reported, e2 occurred whether reported or not.

- Induced causality: One type of event is an input to an event processing component, and the derived event type is the output. This type can be automatically inferred from knowledge of the event processing network.
D4.1: State-of-the-art of event correlation and event processing

- Potential causality: The event $e_1$ is an event that is sent from an event processing network to a consumer $c_1$. The actions of $c_1$ are beyond the borders of the event processing system, but $c_1$ also acts as an event producer and can produce events of type $e_2$. The event processing system cannot know, without further knowledge, whether there is indeed causality among events $e_1$ and $e_2$, but cannot rule out this possibility.

Some studies of root cause analysis at source code level in software engineering have been done, especially in the research of techniques able to provide faster methods to find the origin of the fault, inspired by debugging in real situations of software maintenance.

One example is the work of Zimmermann[87], using repositories like CVS to map bugs from the bug database of Eclipse (one of the largest open-source projects) to their source code locations. The resulting data set listed the number of pre- and post-release defects for every package and file. All data became publicly available, serving as a benchmark for defect prediction models.

Other interesting study is the proposal of Renieris et al.[64], comparing a failing run of a program to correct runs, with the purpose of identifying suspicious parts of the program causing the failure. The method selects the correct run that most resembles the faulty run, according to a distance criterion, compares these two runs, and produces a report of "suspicious" parts of the program. This method is based on the way an "ideal user" would navigate the program using the report to save effort during debugging.

One of the most remarkable contributions to root-cause analysis in software engineering is the work of Zeller[85], which aim is to isolate the variables and values relevant to a failure. The way to achieve this is first considering a failing program as a sequence of changes of program states (events). Applying an algorithm called Delta Debugging to multiple states of the program, it narrows the state difference between a passing run (where the failure does not occur) and a failing run, which reveals a cause-effect chain of the failure. This chain contains the variables and values that caused the failure, by instance: “the failure occurs if and only if variable x has the value y”. If the granularity of the cause-effect chain is increased, the moment where the program state changed to “failure” can be isolated. Hence, this moment in time corresponds to a piece of code, that is the error to be examined. Nevertheless, there is still some work to do on topics like optimization and more case studies.

2.6 Fault Prediction

Event correlation usually refers to dynamic detection of failures (or alarms, alerts, incidents, etc). This means that the techniques are applied to the running systems which events are monitored in order to detect the occurrence of specific situations. An important
aspect of this approach is that correlation can be used in order to detect the occurrence of a specific situation which has not occurred yet. In other words event correlation corresponds in this case to fault prediction. Fault prediction is a well established research topic. However, unlike event correlation, most work on this topic deal with static aspect of the system, i.e. fault prediction is based on static software artifacts such as source code or bug reports instead of dynamic artifacts such as system traces. In this section we give an overview of the type of work being conducted in fault prediction and how some of the results can feed event correlation systems.

A recent survey of fault prediction techniques was provided in [16]. This survey characterizes works on fault prediction according to the type of metrics they use (method level, class level, component level, etc) and the methods they rely on (machine learning, statistics, expertise).

As conjectured in [41], faults in software systems usually do not occur in isolation, but rather in bursts of several related faults. Using a cached history of locations that are likely to have faults, we can predict faults. Those "locations that are likely to have faults" can be found starting from the location of a known (fixed) fault, if we cache the location itself, any locations changed together with the fault, recently added locations, and recently changed locations. Bug occurrences have three types of locality [41] that can be used as fault predictors:

- **Changed entity and new entity localities:** If an entity (attribute/method) was changed recently, it will tend to introduce faults soon. Research shows that methods that changed recently are more likely to be fault-prone than others. In the same way, new entities are more likely to contain faults than existing ones.

- **Temporal locality:** If an entity introduced a fault recently, if will probably tend to introduce other faults soon. Faults are not introduced individually and uniformly over time. When a fault is introduced to an entity another fault will likely be introduced to the same entity soon. An explanation could be that once a fault is introduced, it probably shows an incorrect understanding by the programmer, so it can lead to multiple faults.

- **Spatial locality:** If an entity introduced a fault recently, nearby entities (in the sense of logical coupling) will also tend to introduce faults soon. When programmers make changes based on incorrect or incomplete knowledge, they likely cannot assess the impact of their modifications as well. Thus, when an entity has a fault, there is a good chance of other, nearby entities also having faults. The definition of nearby entities require to define “distance” in terms of logical coupling: If two entities are changed together many times, we give them a short distance, reflecting their logical “closeness”.

The work done with the FixCache and BugCache algorithms analysis [41] by Zimmermann and Zeller reveals that in an evaluation of seven open source projects with more than 200,000 revisions, the cache selects 10% of the source code files and these files account for 73%-95% of faults. At the function/method level, it covers 46%-72% of future faults. The cache can serve as a priority list to test and inspect software whenever resources are limited.

About the cache itself, it can be used as a convenient mechanism for holding the current list of the most fault-prone entities. Instead of creating mathematical functions that predict future faults, the cache selects and removes entities based on the four criteria that we mentioned recently. This way the size of the cache is minimized and it is refined, achieving more accuracy.

The two algorithms based on cache that were evaluated are BugCache and FixCache. The first is based on updating the cache at the moment a fault is not found in the cache. Otherwise, FixCache has a delayed update: when a fault is fixed, the algorithm traces back to the corresponding bug-introducing change, and only then is the cache updated, based on the bug-introducing localities.

Special consideration must be paid in the case of change bursts (consecutive changes in a period of time) in software maintenance. Since every change induces a risk, what could happen if code changes again and again in some period of time? In an empirical study on Windows Vista [57], Zimmermann found that the features of such change bursts have the highest predictive power for defect-prone components. With precision and recall values well above 90%, change bursts significantly improve upon earlier predictors such as complexity metrics, code churn, or organizational structure. As they only rely on version history and a controlled change process, change bursts are straightforward to detect and deploy.

Gruska, Wasylkowski and Zeller [27] propose an approach to parse software engineering projects and learn from them. First they designed a lightweight parser that makes it possible to parse source code of syntactically similar programming languages such as Java, C and C++. This parser mainly capture information about method calls. Then the authors rely on some of their previous work ([80]) in order to extract temporal properties from the parser output. As this approach is lightweight, it can be applied to an important number of projects and be used to determine and rank common temporal properties. The authors applied their approach on 6,000 C projects. Any new C project can then be checked against these properties to detect anomalies, i.e. code that deviates from the “wisdom of the crowds”.

Pan, Kim and Whitehead [60] adopt a different approach and determine common bugs in code by mining configuration management of repositories. In its research, they focused on open source Java project: Eclipse, Columba, JEdit, Scarab, ArgoUML, Lucene, and
MegaMek. By learning from these repositories, they determine what are common bug fixe patterns. They discovered for instance that the must common bug fix patterns are “method call with different actual parameter values” and “change in if conditional”. From these patterns, they infer what common bugs are related to dealing with all the possible cases of method parameters and introducing wrong checks in conditionals. These results indicates what part of the code may be prone to errors.

Finally, we point out here the emergence of some works dealing with fault prediction at runtime (e.g. [15, 83]). In these works, snapshots of the different components of the system (which can be distributed) are collected and used for model checking. This approach provides a trade-off in terms of complexity as applying model checking on snapshots at runtime reduces the possible value domains of the variables, hence the complexity. By performing automatic decentralized verification at runtime, the system is able to predict possible future faults occurrences. In the same spirit, Java PathFinder (see e.g. [78]) makes it possible to explore the different branches of a program flow at any given point at runtime. From a given execution of a program, some inputs are automatically generated and the program is executed until all the relevant branches have been visited. If one of these branches led to a fault, it can be detected.

These example of research studies in the area of fault prediction show that the trend in this domain is either to consider static artifacts of the software in order to help locate its parts that are more likely to lead to a fault; or to consider automatic testing or verification at runtime. Although this later approach is very promising, it induces a non neglectable performance overhead on the system.
3 Event Correlation Techniques

In this chapter we will perform a first approach of the general existing techniques of event correlation, prior to the overview of the existing technologies and systems that use these techniques. A lot of techniques are available, and combinations of them, but there is no one better than others, it depends highly on the problem at hand [52], so we will research the usability domain of the most common techniques, always keeping in mind the kind of events and the strategy needed to identify the main software maintenance situations.

3.1 Main Event Correlation Techniques

3.1.1 Rule Based Event Correlation

In rule based event correlation (RBC), the system constantly uses a set of predefined rules to evaluate incoming observations until a conclusion is reached. Therefore the correlation ability depends solely on the depth and capability of the rule set.

In a rule based event correlation engine, information is represented in three levels [55]:

- Knowledge level: Domain-specific expert information is available in a knowledge base, which is the rule repository.

- Data level: Information about the problem at hand is allocated in the working memory. At this level, facts are stored.

- Control level: The response about how to apply the rules from the knowledge base to solve a given problem is located at the inference engine.

**Application domain of rule-based event correlation** Let's have a look at the properties of systems based on rule-base correlation, so we can identify the kind of environment this technique is suitable for. As we have seen, control and knowledge are separated, so knowledge can be updated without changing the program code of the engine. This knowledge traditionally relies on the domain-specific expertise of an engineer, and in most cases, it has to be entered into the system manually. Hence, this procedure is time consuming and frequent changes make it a tedious maintenance system. Learning and automation of the knowledge updating is difficult for pure rule-based systems, so they are prone to
D4.1: State-of-the-art of event correlation and event processing

fail when new or unexpected situations occur. In case some unforeseen problem occurs and it is not covered by the set of rules, the knowledge engineer has to develop new rules of coverage. Thus, if the domain of coverage changes too rapidly, it becomes difficult to maintain an accurate set of rules [52]. Another problem is that as a set of rules gets larger and more complex, there is an increased chance that two rules can cause inconsistent results. For these reasons, most knowledge engineers advise that RBC systems are best used in domains that are well-known and not expected to change too much over time. Nevertheless, RBC engines based on the RETE algorithm [68] are less sensitive to problem changes, since they can continue the correlation process without redoing the correlation path already performed. However, in case the change is too fast, RETE loses its effectiveness.

3.1.2 Case Based Event Correlation

In Case based correlation (CBC), the correlation system tries to solve a given problem by searching for the most similar case from a case library and retrieve the solution. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived. Sometimes it may require the adaptation of a solution stored in the library in order to solve the current problem, to propose a new solution. This principle consists in solving incremental problems with a sustained learning component. After applying the new solution to the problem, the outcome will be verified and if it is successful, the new case (problem and solution) will be stored in the library. Otherwise, a better solution must be proposed, which will be verified and incorporated to the library. Hence, a CBC system learns from experience and can adapt to unknown problems.

Application domain of case-based event correlation CBC offers several distinct operational features:

- First, a case in CBC can be a semantically rich data structure, thereby making it ideal for management tasks dealing with complex problems. As the practice of CBC has demonstrated, a complete problem can be solved by recognizing only one case, which reduces the problem resolution cycle considerably.

- Second, the case adaptation algorithm used in CBC makes this approach more suitable for solving problems where the exact solution either does not exist or is too costly. In this sense, CBC systems are less rigid than rule-based correlation systems.

- Third, being by nature a learning system, CBC allows the operational behavior of the event correlation process to be improved without additional hard-coding.
Since solutions for the current problem can be derived from working solutions to past problems, these can be presented as evidence, so it can increase the user acceptance. Nevertheless, the retrieval of similar cases must be done carefully, since it can lead to solutions that do not fit the current problem.

Originally, CBC was not intended for large-scale real-time applications. Due to the relative complexity of the cases, the case-matching procedures in the majority of CBC applications are weaker than the ones implemented in Codebook systems or in RBC systems based on the Rete algorithm. Certain performance improvements could be achieved by simplifying the structure of the cases. Another limitation concerns temporal correlation capabilities, since no CBC implementation has reported a capability of temporal correlation.

Finally, we should mention the similarity between a case library and a ticketing system. In case of software maintenance, a CBC system may suggest to use a ticketing system as case library. This idea has been put in practice in a trouble ticketing system called CRITTER [55]. The results gave the opportunity to rate solutions and give feedback from the users.

3.1.3 Finite State Machine Based Correlation

Finite State Machines are popular in computer science for their power and simplicity. They make it possible to describe and analyse states and behaviors of a system. Extensive definitions and studies related to Finite State Machines can be found in [35]. We provide here a definition of Finite State Machines (FSM). An FSM is a 5-tuple \((\Sigma, Q, q_0, Q_m, \delta)\), where \(\Sigma\) is a finite alphabet (set of events), \(Q\) a finite set of states, \(q_0 \in Q\) is the initial state of the FSM, \(Q_m\) is a set of marked (final) states and \(\delta : Q \times \Sigma \rightarrow Q\) is the partial transition function. Intuitively, for a sequence of events \(s \in \Sigma^*\), \(s\) is a possible behavior of the system if \(\delta(q_0, s)\) is defined. If it is, \(\delta(q_0, s)\) represents the state the system reaches after the sequence of event \(s\) occurred. If this new state is marked, it usually means sequence \(s\) corresponds the completeness of a task. The set of behaviors of system \(G\) is the language generated by its FSM. An important aspect of FSM is that they are finite and usually compact. Therefore, they allow for effective and efficient computations and analyses. Although the FSM complexity of nowadays systems makes it difficult to model them in details, some of their aspects can always be modelled with some abstraction.

In the field of network management, event correlation is sometimes called alert (or alarm) correlation. The main goal of alert correlation in a network management system is to localize the faults that occur. Many solutions to this problem have been considered, based on several field of computer science such as artificial intelligence, graph theory, neural networks, information theory, and automata theory. For instance, Finite State
Machines represent a type of models which can be applied to model-based alert correlation approaches. Deep knowledge of the system may describe its structure (static knowledge) and function (dynamic knowledge). Finite State Machine have been considered in some work on diagnosis of Discrete Event Systems (DES). Machines called diagnosers are automatically synthesised and are able, under some conditions, to detect the occurrence of unobservable failures at runtime. Failure Diagnosis of DES relies on a model of the system to be diagnosed and aims to automatically compute a model of a diagnoser. This diagnoser is able to detect faults at runtime, when the failures are assumed not to be observable. The goal of the diagnoser is to infer the presence of failures from the sequence of observed events. The diagnoser is a Finite State Machine that is built from the model of the system, which is also assumed to be a Finite State Machine.

At runtime, the diagnoser observes the behavior of the system and estimates the state which it has reached. The diagnoser possesses information about the possible failures in the different states of the system and can detect them in a finite time. Figure 3.1 illustrates how FSM can model both the system and a diagnoser. In this example, an FSM representing the behavior of the system is given on the left of Figure 3.1. Its alphabet is \(\{a, b, c, d, e, \sigma\}\) and it is assumed that \(\sigma\) is the only unobservable failure and that both \(b\) and \(\sigma\) are unobservable. The FSM given on the right of Figure 3.1 represents a diagnoser for \(G\). \(F1\) denotes the type of failure \(\sigma\) and \(N\) denotes the type of a normal event. The meaning of the diagnoser can be expressed as follows: if for instance the diagnoser is in state \(6N, 7N\), then as \(b\) is unobservable it means that after observing events \(a\) and \(c\) the system is either in state \(6\) (corresponding to a normal behavior) or in state \(7\) (also corresponding to a normal behavior). Then, if event \(d\) is observed, the system either enters state \(8\) (which corresponds to a normal behavior) or enters state \(10\), which, according to the diagnoser, corresponds to the occurrence of a failure of type \(F1\). Therefore, this example illustrates how FSMs can be used to monitor a system and conclude on the occurrence of unobservable failures. The diagnoser can be computed in an automated way under some conditions. These conditions and algorithms are presented in [67].

**Application domain of finite state machine based correlation** An important benefit of FSM based event correlation is that it makes it possible to formally define and automatically compute an FSM representing a diagnoser. The price to pay for this approach is unfortunately quite high as it requires that an FSM modelling the behaviors of the system is available, which is unfortunately rarely the case in practice.

### 3.1.4 Model Based Correlation

Model Based Correlation refers to the use of a model of the physical world representing the structure and behavior of the system under observation, as an inference method.
This approach does not suggest a detailed technique, but a paradigm instead. It has a connection with rule-based systems, since a practical implementation might use a rule-based model, but it differs from RBC since it specifies a system model, with events as consequences of certain model states and transitions, while RBC specifies event patterns as conditions for certain actions. FSM systems are closer to the MBC approach than RBC systems are.

**Application domain of model based correlation** In software maintenance, the use of MBC methods may be unsuitable, since the description of system, application and user and the behavior of all of them would be really difficult. This approach would be suitable to fault diagnosis in an electrical circuit, for instance, since the structure is specified as a circuit diagram and the behavior is defined by a few rules.

### 3.1.5 Probabilistic Event correlation

Probabilistic event correlation emerges from the idea that in any practical domain, there is an element of uncertainty, probably due to a lack of confidence in the way the reality is modeled or due to incomplete data. Hence, an event is subject to a probability index (ranging between zero and one), instead of claiming that the event is absolutely true or false [52].

![System G and Diagnoser $G_d$.](image)
Some works considering system intrusion and attacks use a different terminology and call “alarms” or “alerts” some relevant observable events of the system. In this case, alarm/alert correlation corresponds to event correlation where conclusions on the occurrence of unobservable events can be made from observations of some observable alarms. In [77], the authors consider a probabilistic approach in order to correlate security alerts. Alerts are assumed to possess attributes which are used to measure similarities between alerts. The correlation approach uses similarity metrics to group alerts into higher level alerts. Probabilities are used in order to relax some similarity requirements of the alert attributes.

In [73, 44, 72], Steinder et al. consider probabilistic approaches to tackle “fault localization”. The notion of alarm is also present in this work where fault localization refers to fault isolation, alarm/event correlation, and root cause analysis. All these cases corresponding to analysing a set of observed fault indications in order to give meaning to alarm occurrences. In [73] for instance, probabilistic symptom fault maps are used to model fault propagation. The approach consists of determining the most likely set of faults through incremental updating of a symptom-explanation hypothesis. When symptoms are observed, several hypotheses are proposed and ranked according to some relevance measure. With this technique, the occurrence of several simultaneous independent faults can be detected.

In [44], the authors survey different techniques for fault localization, some of which consider Belief networks. A Belief network can be represented as a directed acyclic graph whose nodes represent random variables and edges represent influences between the variables corresponding to the involved nodes. For event correlation purposes, the random variables represent the occurrence events. The fault localization problem may be formulated as a problem of calculating the most probable explanation to the observed events. Although the problem is known to be NP-hard in general, Belief networks are used to answer this problem in works such as described in [61]. In [38] the author uses theories centering around artificial intelligence in order to perform fault management. The proposed approach consists of constructing a belief network for each particular fault to be detected. Then these small networks can be combined in order to provide a complete belief network. A difficulty with this process lies in the probability assignment step, since each relationship in a belief network has some probability associated with it. If the probabilities are too different from the correct values, the network representation will be, at best, misguided.

Other works on fault localization using belief networks have been conducted such as for instance [79, 34, 18]. However, these works are restricted to quite specific fault diagnosis problems, using simplified belief network.

Other type of probabilistic models were also considered in [81, 76], neural networks are
considered for event correlation. As explained in [81], neural network approach possess some advantages. First, unlike Belief network, they can be trained and no expert knowledge is required for it. Moreover, modifying the model can be done relatively easily when new knowledge become available. Finally, neural network have the property of being quite resistant to noise.

Gao, Gaudin and Sterrit [23, 24, 74] considered other type of probabilistics models called Bayesian Network. The following example, taken from Gaudin [24] illustrates this type of models. Figure 3.2 represents a Bayesian Network for diagnostics of WiFi connection issues. A Bayesian Network can be represented as a graph. Each node corresponds to a variable and each edge represents some dependencies between the variables. A conditional probability table is associated to each node. For instance, Table 3.1 represents the conditional probabilities associated to node “Page Not Displayed”. From this model, given some observations, it is possible to automatically infer the probability of the values of all the unobserved variables. Nodes “Access Point Issue” and “Server Issue” are called “targeted issues” or, more frequently “issues” in the rest of this document. All the other nodes are called “observation”.

For example, when in the range of a WiFi access point, the requested page is not displayed. Being in the range of an access point as well as the failure to display a requested page are easily observable. A remote server being down is not as easy to observe but can

Figure 3.2: An example of Bayesian Network for WiFi Connection Issues
D4.1: State-of-the-art of event correlation and event processing

<table>
<thead>
<tr>
<th>In range of Access Point</th>
<th>Low Connectivity</th>
<th>Page Not Displayed = False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>0.98</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>0.32</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 3.1: A possible conditional probability table for node “Page not Displayed”.

be inferred by the model from observations and thanks to past experiences.

Unlike neural networks, the graphical representation and the structure of Bayesian Networks makes them easy to read and manually modify/augment. Moreover, several tools exist with which it is possible to learn Bayesian Networks from data: Banjo\cite{29}, Genie\cite{19}, Weka\cite{82}, etc.

In \cite{45} the author proposes to perform event correlation using a combination of case based correlation, rule based correlation, model based correlation, Bayesian networks, and neural networks. Finally other works such as \cite{26} correlate security events using Bayesian correlation in order to automatically detect intrusion.

**Application domain of probabilistic event correlation** Bayesian correlation is the most popular example of probabilistic event correlation. Regarding its usability, most of the times developers notice the difficulty of coming up with prior probabilities before computation begins, which is a requirement for this kind of systems. As a consequence, bayesian correlation only works well in cases where such statistical data are available and an expert can come up with accurate prior probabilities, conditions that are not frequently met.

### 3.1.6 Codebook Based Event Correlation

With the same flavour as the rule-based approach, the codebook is somewhat similar, but always producing a diagnosis. Kliger et al\cite{43} describe this system, where the aim is the localization of problems and the method is the selection of a suitable subset of symptom events associated to their underlying problems. This subset of symptom events is the codebook, and for each problem, a binary vector is created, indicating whether each symptom in the codebook can be caused by that specific problem. With the purpose of identifying problems, the events that are contained in the codebook are monitored in real time, and all together they form the event vector. When some events occur, the event vector and the vector for each problem are compared. The vector with the smallest Hamming distance to the observed vector is selected, identifying the observed problem.

To illustrate this process, consider the following example: Consider a problem A, which causes symptoms X, Y and Z, problem B, which causes symptom W and Y and problem
**D4.1: State-of-the-art of event correlation and event processing**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.2: Example of codebook correlation matrix

C, which causes Y and Z. Since symptom Y is caused by all problems, it does not provide any information, so the codebook is represented with the following table:

If the events W, X and Z occur, the problem vector with the smallest Hamming distance would be problem A, so the problem would be identified.

**Application domain of codebook based event correlation**

The codebook approach needs the same expert knowledge as the rule-base scheme in order to populate the codebook properly. The missing notion of time is also one of the lacks of this technique. When a set of event is said to have occurred together, there is no information about the time window applied to group the events. Neither the event order is available, so all events are assumed to have occurred simultaneously. The identification of pattern, where the information regarding the order of occurrence is a must, is an unavailable feature when using this technique. In addition to that, the fact that events do not have any properties associated to each of them is another significant problem in cases where relationships between events have a remarkable significance.

**3.1.7 Comparison of the Main Event Correlation Techniques**

Müller’s comparison of several event correlation approaches gives a list of some of its pros and cons. The summary of this comparison is shown in the following table:
D4.1: State-of-the-art of event correlation and event processing

<table>
<thead>
<tr>
<th>Technique</th>
<th>Pro</th>
<th>Contra</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM</td>
<td>Simple, good as a basic model, easy to understand</td>
<td>Too simple for practical applications, no tolerance to noise</td>
</tr>
<tr>
<td>RBC</td>
<td>Transparent behaviour, close to natural language, modularity</td>
<td>Time-consuming maintenance, difficult to learn from experience</td>
</tr>
<tr>
<td>CBC</td>
<td>Automatic learning from experience, correlation from past experience is natural</td>
<td>Automatic solution adaptation and reuse is difficult</td>
</tr>
<tr>
<td>MBC</td>
<td>Relies on deep knowledge</td>
<td>Description of behaviour and structure may be difficult in practice</td>
</tr>
<tr>
<td>Codebook</td>
<td>Fast, robust, adapts to changes</td>
<td>Description of behaviour manually is tedious; no notion of time</td>
</tr>
<tr>
<td>Bayesian networks</td>
<td>Good theoretical foundation</td>
<td>Probabilistic inference is NP-hard</td>
</tr>
</tbody>
</table>

Table 3.4: Comparison of existing techniques [55]

3.2 Semantic Event Correlation

Complementing the main reported event correlation techniques explained in the last section, and looking at a higher level of abstraction, there is one more discipline which is growing in the event correlation field that we cannot obviate, which is the semantic event correlation. We have not presented this technique in the same level as the others, since it can be understood as an evolution of them, and from a certain point of view, a hybrid technique of model-based correlation and rule-based correlation.

So far, traditional event correlation has been limited to match only syntactically exactly equal values of event attributes to decide whether or not two events are related. However, this approach is adequate only with a high data quality of the attribute values on which the correlations are based. While this is even hard to ensure for a single organization, it takes a lot of time and effort to allow for the required data quality in events coming from organizations that use different terminologies to describe their business data.

Semantic technologies are usually associated with ontologies. An ontology is a formal, explicit and unambiguous representation of knowledge as a set of concepts within a domain. They have various advantages, such as making domain knowledge explicit, and sharing, using and reusing this information between people and software agents.

In the context of semantic event correlation, ontologies provide an explicit way to model these differences in a semantic layer that is decoupled from the correlation, which facilitates better adaptability and reusability for both: the semantic model and the event correlation engine. Furthermore, semantic modeling enables building correlations based
on inherited meanings of terms as well as on relationships between them [53].

Moser et al. [53] described three application scenarios where some events could not be correlated with traditional syntactic correlation, because of semantically heterogeneous terminologies, and with the use of ontologies, they could be correlated based on meaning, inheritance and relations. As explained in the first use case of [53], ontologies offers indeed a very attractive framework to map concepts coming from heterogeneous systems. Ontologies have other interesting features. Next, we define and illustrate some of them.

To describe the elements of an ontology it is helpful to have a certain example domain in mind. In this case, we describe the domain of pets and their owners, inspired by the ontology used in [36]. An ontology consists mainly of the following parts:

- Individuals (also called Instances) denoting the objects in a domain. In our example John, Mary and Fido could be individuals.
- Classes (also called Concepts) that are sets containing individuals. In our example Person, Owner, Pet, Dog and Cat can be classes in a domain.
- Properties (also called Slots or Relationships) that are binary relationships between individuals. The fact that John has a pet called Fido is indicated by the property hasPet(John, Fido).

For example, John and Mary are instances of the class Person. Both classes and properties can have hierarchies. A class hierarchy for example, can show that the class Owner is a subclass of the class Person. Similarly, there can exist a hierarchy between properties: for example the hasDog property can be seen as a subproperty of hasPet.

We can also impose certain restrictions on classes and relationships. In our given example, we could disallow an individual to be both a Person and a Pet. A property, for example, can have domain and range restrictions on its parameters. In the case of the property hasPet, it could have a restriction that its domain and range apply to Owners and Pets respectively. There are also many other possible restrictions, such as cardinality, transitivity, etc. In terms of event correlation, such restrictions are mentioned in the third use-case of [53], as they allow to express conditions under which some events should be correlated.

On top of the modeling facilities already mentioned and the Ontologies offer, there are other interesting features that Ontologies possess (see e.g. [59]):

- Being both human and machine readable: ontologies can be created by humans using powerful editors such as Protégé ([36]). As they rely on some formal semantics, some automation of their processing can be performed by programs.
- Enabling reuse of domain knowledge: once an ontology for a certain domain is created, other users can incorporate this knowledge into their own works and even
ontologies. For example, if a general ontology is created regarding time and its concepts, then other applications and ontologies can use that ontology instead of implementing the domain knowledge for themselves.

- Making explicit domain assumptions and separating the domain knowledge from the operational knowledge. Once knowledge of a domain is separated from the rest of the application, making changes to either of them separately is much easier than if they were interwoven. For example the people updating the domain knowledge, might be experts in that domain, such as medical knowledge, but not in the operational details around it like software engineering and vice versa. Separating the two types of knowledge makes it easier for them to do their job, without being bogged down in domains outside their expertise.

- Analyzing domain knowledge. If the domain knowledge is separated and defined, formal analysis on it can be applied. This allows us to improve upon that knowledge, as well as derive previously unknown facts from the knowledge.

An ontology language is a formal language used to encode the ontology. The semantics of an ontology language are often based on Description Logics (DL). Description logics are used to give a formal logic based semantics to knowledge representation systems, while keeping in mind the balance between efficient correlation and expressiveness [11]. There are many different kinds of descriptions logics [11]. There exist a number of common correlation tasks for ontologies that can be automated (see e.g. [8]). In terms of event correlation techniques, these correlation capabilities could be used to automate and ease the management of event correlation rules.

- Class membership: Deducing if an object is an instance of a class. For example if John is an Owner and Owner is a subclass of Person than we can deduce that John is a Person, as this statement will be in all the models of the ontology. This idea is related to the second use-case described in [54] where the class hierarchy can be used to automatically infer that some events should be correlated although they have not been explicitly described as such.

- Classification: Deducing all subclass relationships between classes. For example given that we know that a DogOwner is a subclass of Owner and that Owner is a subclass of Person we can infer that DogOwner is a subclass of Person. In case of event correlation for attack detection, this idea makes it possible to automatically determine a hierarchy of possible attacks.

- Equivalence of Classes: Deducing if two classes are the same, by checking if they have the same extensions. For example if the class DogOwner is equivalent to
D4.1: State-of-the-art of event correlation and event processing

CanineOwner, and CanineOwner is equivalent to HoundOwner than DogOwner should be equivalent to HoundOwner. In terms of event correlation for attack detection again, this idea would make it possible to identify two types of attacks, that may possess different names but that are characterized by the exact same event correlation rule.

- Consistency of a Class: Class definition and relationship with inherited classes must be consistent in meaning, attributes and functionality. For example if we define the two classes Male and Female disjoint, any class that is the subclass of those two classes will be inconsistent. In terms of event correlation for attack detection again, this idea would make it possible to detect inconsistencies in the types of attacks that are described.

There are various reasoners that can perform these inferences, and more, on an ontology with a decidable semantic, such as Pellet [69]. Such correlation facilities are of a great help when designing and using ontologies. As it was previously mentioned, doing these tasks efficiently has a tradeoff with expressiveness, and there is a great deal of ongoing research on how to manage this balance [11].

Several types of ontologies have been defined and studied over the years. Among the most well known ones, we can mention RDF, DAML+OIL and OWL.

An important feature of ontologies are that they ease correlation with knowledge representation. However, the ease of correlation always comes with some constraints on the knowledge expressiveness. This balance has to be taken into account when choosing an ontology language for a certain application. The OWL family of languages, both OWL1 and OWL2 offer various choices in this. They are also by far the most commonly used as well as a current W3C standard, means they are an excellent choice for creating ontologies.

In software engineering, systems and information about them diverge quickly in time, resulting in difficulties understanding and maintaining them. This divergence is typically a consequence of the loss of coupling between software components and system metadata. Hyland-Wood [39] proposed a methodology for capturing and making use of system metadata, coupling it with information regarding software components, relating it to an ontology of software engineering concepts (SEC ontology) and maintaining it over time. This methodology is based on standard data representations and may be applied to existing software systems. It would be robust in the sense that most of the required information may be automatically generated from existing sources, thus reducing the need for human input. We have not found any evidence of a real SEC ontology, and at this point, it seems to be an idea to be evolved and enriched along time.

In addition to that, the level of advance of semantic event correlation technologies at present do not seem to provide a complete implementation using ontologies. A recent
study from Zhu et al.\cite{86} reveals that one of the main event correlation techniques shown in the present document, the rule-based correlation, is characterized by the use of non-standard rule formats that are not compatible with semantic web standards. This paper also describes an approach to define an event ontology, using Semantic Web Rules Language to encode domain knowledge about event relationships, with the aim of developing a prototype for automatic event correlation, based on domain knowledge updates.

### 3.3 Main Issues

This section aims to pinpoint the most remarkable issues of the event correlation field at present time. Gathering this information will provide a good point of view, emphasizing our research regarding associated techniques that can lead us to advance in the right direction, to be able to cope with these issues.

**Incomplete knowledge** Event correlators should not assume that they have complete knowledge, but they have to reason instead with incomplete and sometimes inaccurate knowledge\cite{52}. In software maintenance, it will not be different since it is a wide area, and knowledge is far from being complete and accurate.

Among the reasons of this incomplete knowledge, we should mention the limitation of network bandwidth, since the amount is usually limited by administrators. In addition to that, the event correlator cannot keep an up-to-date knowledge of all the managed elements in its domains in real-time, especially when it is physically distributed, so it usually works by sampling.

As a result of this incomplete knowledge, event correlation can be affected, obstructing the maintenance of knowledge and making wrong inductive inferences because of the imperfect data used for correlation. Some platforms try to avoid this problem using complex event correlation models that can deal with this incomplete knowledge of the domain, and in some cases with the aid of probabilistic event correlation.

**Unrealistic assumptions** Some of the algorithms used in integrated management for correlating events are frequently hold on some assumptions, in order to simplify the problems. For example, network management algorithms usually apply in assuming the a stable and static network topology. These assumptions might be not real in highly dynamic environments, hence making necessary the use of other algorithms able to work efficiently when these assumptions fail.

**Insufficient automation** At present, one of the properties of event correlation is the lack of automation in the sense of learning from experience and adapting to un-coded
situations occurring in the systems. Most of approaches described in the section of existing
techniques (rule-based, probabilistic, model-based,...) are characterized by this lack of
automation[52], covering perfectly small, unchanging and stable systems but suffering
from adaptation to larger and more dynamic management domains.

In that sense, some of the research that the present document will try to cover is the
use of learning to event correlation, as a related concept that will be presented next.

3.4 Learning

One of the main issues in event correlation identified by [52] is the lack of automo-
tation, meaning that event correlators are configured manually and patterns to be detected are
defined by human experts. This procedure is time consuming and does not scale well
in dynamic and complex systems, limiting the applicability of event correlators to small,
static domains. Machine learning techniques can be used to learn patterns and relation-
ships among events from training data instead of specifying them manually, they help save
time and give event correlators the ability to dynamically adapt to changing environments
and enable the usage of event correlators in dynamic and complex environments., In this
section, we investigate different approaches of learning for several types of correlation
techniques. The general idea is to use machine learning techniques in order to learn the
structure of patterns from data. The learned pattern structure can be represented in
different ways like rules, cases or Bayesian networks and can be used in event correlators
and replace or expand the pattern definitions from human experts.

3.4.1 Mining Association Rules to Find Patterns and
Relationships between Events

Association rules represent associations between items in the form of rules. Historically,
they have been developed in the area of market basket analysis and are used to find items
that are frequently bought together. They can be adopted to events to get insights into
relations between events and to find out which events frequently occur together.

Association rules operate on sets of items that usually originate from buying transac-
tions. An association rule consists of an antecedent and a consequent part where each
of those parts is a collection of items. The rule expresses the fact that if all items of
the antecedent are present in a transaction then the probability is high that the items
in the consequent part of the rule are also part in the transaction. An example of an
association rules is \{milk, bread\} \implies \{butter\} (or referring to software engineering:
\{debugger, reporter\} \implies \{bugreport\} ), denoting the fact that if a customers buys milk
and bread she is likely to also buy butter.
Algorithms that find association rules such as the Apriori algorithm operate on sets of transactions where each transaction consists of a number of items. The algorithm builds possible item sets recursively, starting from item sets with one item to item sets that contain all possible items. From each of these item sets association rules are derived by making a subset of the item set to the antecedent of the rule and the remaining items to the consequent. In order to focus search and obtain meaningful association rules, item sets and association rules are judged based on their support and confidence. Support denotes the number of transactions in which all items of a rule (all items of the antecedent and all items of the consequent) are present and confidence denotes the fraction of transactions which the rule judges correctly divided by the number of transactions in which the antecedent of the rule applies. Item sets and association rules that do not match minimum criteria for support and confidence are ignored.

Association rules can be applied to events by looking at events instead of items. Given sets of events (obtained e.g. by event bursts that occur in a given time frame) an association rule mining algorithm can be applied and the resulting association rules will give information about which events frequently occur together. Disadvantages of using association rules in event correlation are that they do not consider orders between events and that a mechanism how to obtain sets of events from an event stream or an event cloud has to be found. Due to these disadvantages, association rules might be of limited use for direct application in event correlation engines but can be used in order to get insights into relations between events and as basis to define event patterns like rules.

The Apriori algorithm mentioned in this section was developed by Agrawal et. al. [4, 5]. A survey of association rule mining can be found in [17] and [33]. Brin et. al. [13] extend association rules to cover correlations also and Luo et. al. [49] combine association rules and fuzzy logic in order to mine rules for intrusion detection. Klemettinen et. al. [42] use association rules to detect anomalies in telecommunication systems and raise alarms based on it. Stolfo et. al. [75] use association rules to analyze raw data and to find patterns in the area of intrusion detection.

3.4.2 Learning Sequential Patterns

Association rules do not consider the order of events they are operating on. In contrast, sequential patterns do consider the order of events.

Mannila et. al. [50, 30] describe algorithms that are used to recognize frequent episodes in event sequences. They define an episode as “a partially ordered collection of events occurring together”. For example, an episode can be parallel (events that occur in parallel with no imposed order) or serial (a total order is imposed between all events). Event sequences are streams of events where each event has an associated point in time. The
D4.1: State-of-the-art of event correlation and event processing

algorithms proceed similar to the algorithms used for mining association rules: They start with small sub episodes and extend the size of the episodes under consideration if all sub episodes have a certain frequency. This is analogous to the item set generation for association rules. Rules can be generated easily from a frequent episode by taking sub episodes as antecedent of the rule and the full episode as the consequence. Pruning can be done by specifying minimal thresholds for confidence and support.

Sequential pattern mining deals with mining of frequently occurring ordered events called patterns and was first introduced by Agrawal et al. [6]. Let \( I = \{i_1, \ldots, i_n\} \) be a list of items. All subsets of \( I \) are called item sets. A sequence \( s = \{t_1, \ldots, t_m\} \) with \( t_i \subseteq I \) is an ordered list of item sets. If a sequence is contained in more sequences in a sequence database as specified by a minimum threshold property it is called a frequent sequential pattern.

The GSP algorithm introduced in [71] algorithm works very similar to the Apriori algorithm in association rule mining: It generates candidates of frequent sequential patterns by starting with short sequences and producing longer sequences by combining short sequences. Sequences that do not have the required support are ignored. GSP also includes time constraints, a sliding-time window and user-defined taxonomies for items.

Another mining algorithm for sequential patterns is PrefixSpan [62]. In contrast to GSP, PrefixSpan does not create candidate sequences but proceeds in a divide and conquer manner: It first finds all patterns of length 1. These patterns are used as prefixes of sequences and the search space is partitioned according to the prefixes. Frequent sequential patterns with the corresponding prefix are searched in each part of the search space. Advantages of such an approach is that only frequent sequential patterns are created in such a way and the expensive candidate generation is not needed. Consequently, PrefixSpan outperforms GSP in most cases as shown in [63].

Rules can be generated easily from sequential patterns as described in [46]. For example, a sequential rule of form \( X \implies Y \) with \( X \) sub sequence of \( Y \) can predict (with certain accuracy) that, having observed the sequence \( X \), it is likely to see \( Y \) (the missing events in \( Y \) that have not yet been observed). Another example for a rule is a class sequential rule that assigns classes to event sequences based on sequential patterns that occur in the sequence.

Han et. al. [28] investigate the current state of sequential pattern mining in depth and cover also historical approaches and further directions.

3.4.3 Learning Patterns for Rule-based Correlation

When considering rule-based correlation two problems come into play: the need of expert knowledge in order to create the rules and the need to update the rules in changing
systems. Machine learning techniques can help to automate the process of creating and maintaining rules. When considering events in software maintenance the time in which they occurred needs to be considered; in some cases, the duration of the event is also important. Due to the fact that FastFix is mainly concerned with software maintenance and therefore will handle time events, in this section we focus on machine learning techniques that take temporal constraints into consideration.

Rule discovery

Rules can be discovered from temporal data through modifications of the Apriori algorithm explained in Section 3.4.1. Traditional association rule algorithms can be extended in order to include temporal constraints, as done in [7, 40]. The presence of temporal association rules is subject to three interpretations [65]:

- The earlier event plays a role in causing the later event.
- There are third events or sets of events that cause two other events.
- The time overlap between both events is coincidental.

Another possibility for rule discovery involving temporal data is the mining of sequences of events as described in Section 3.4.2.

Maintaining rules

A main challenge in rule-based correlation is rule maintenance. This has been the focus of work realized by Abraham et al. [2], Spiliopoulou et al. [70] and Horschka et al. [37] which concentrate on mining time-stamped rules in order to find changes in the rule. This type of work corresponds to the “Higher Order Mining” field and consists on the application of data mining mechanisms over existing mining results. Through these mining mechanisms changes in the rule structure, such as the increase or decrease of the support and confidence, as well as the need to add an extra term in the left part of the rule can be deduced. The terms support and confidence where proposed by Agrawal et al. [4] and can be used for evaluating a rule’s relevance [65]. Support is the extent to which the data is relevant (either positively or negatively) to the rule, whereas confidence is the extent to which, within those that are relevant, the rule holds true. There are some application were these metrics can be misleading [13] and other metrics have been proposed [12].
3.4.4 Learning for Case-based Correlation

Learning for case-based correlation or case-based correlation is different compared to the other correlation approaches that have been investigated in this report. In case of rule-based, codebook-based or probability-based correlation, first a event or pattern structure is either specified manually or learned from data. This structure is expressed as a rule or a Bayesian network and is used when performing event correlation. In contrast, in the case of case-based correlation no structure is specified or learned in advance. Instead, a set of problems and corresponding solutions is formalized in some way and stored in a case database. When a new instance of an event or a stream of events has to be correlated, the most similar case from the case database is searched and its solution is applied to the current instance - either unmodified or adapted to the current situation. From a learning perspective, this moves all processing and correlation to the time when an instance has to be correlated.

The main challenges of a case-based correlation approach are how to compute the similarity between two cases (which is domain dependent), how to efficiently find the most similar case, which of the cases to store in the database (there is a trade-off between accuracy/broad coverage and speed/database size) and how to apply the solution of one case to the current instance.

An approach often used for case-based learning is called k nearest neighbour. In this approach, the k nearest neighbours of an instance to be correlated are retrieved from the case database and the solutions of these k nearest neighbours are combined and applied to the current instance. To determine the distance between two cases, corresponding metrics have to be specified and this usually requires normalization of all dimensions in order to make them comparable. The most common application example is classification in which the current instance is assigned to the class obtained by a majority vote of the k nearest neighbours. The k nearest neighbour approach is discussed in [66] and [82].

3.4.5 Learning Patterns for Probability-based Correlation

As stated by [52], there is a certain degree of uncertainty involved when performing event correlation for real systems that is caused by uncertainty of data (faulty event sensors, ...) and incomplete knowledge (some events missing, ...). In order to model such uncertainty, probability-based methods are usually used. In this section we focus on learning Bayesian networks that were introduced in Section 3.1.5.

As already explained, a Bayesian network is a directed acyclic graph with variables (in the case of event correlation events) as nodes and an edge from node A to node B if B is conditionally dependent on A. At every node, a conditional probability distribution table is stored which represents the probability of an event dependent on the occurrence of its
parent events. This structure enables modeling causal dependencies between events in a probabilistic way. Additionally, not directly observable variables, like the classification of an event, can be modeled as a node without parents and directly observable variables like event features are modeled as nodes that conditionally depend on the variable not directly observable. When the conditional probability tables at every node are known, the probability of the unobservable parent node can be inferred based on the real observations of the observable nodes. A normal Bayesian network can be used to classify single events whereas dynamic Bayesian networks (such as hidden Markov models) can be used to classify sequences of events and incorporate the notion of time. Examples for the usage of Bayesian networks in event correlation can be found in Section 3.1.5.

The task of creating a Bayesian network consists of two steps: the first is to construct the structure of the network which boils down to defining all variables or events that are important and to specify causal relationships between them. The second step is to specify the conditional probability distributions or parameters for each node. Both steps can be automated by learning and will shortly be examined. Learning of Bayesian networks is treated in depth by Russel et. al. [66], Heckerman [31] and Neapolitan [58].

Learning the probability parameters of a pre-defined Bayesian network structure can be done using statistical methods and is relatively easy if all variables can be observed, there is no incomplete knowledge and the parameters are assumed to be independent from each other. If these conditions are not met, techniques have been proposed in order to deal with each of these issue. For example, unobserved parameters have to be estimated using maximum likelihood or expectation maximization techniques and incomplete knowledge of samples can be mitigated using approximation schemes like Monte Carlo techniques or Gaussian approximation.

Learning the structure of a Bayesian network is usually approached by starting from an existing structure (in the simplest case the structure with all variables as nodes and no edges between them) and trying to improve the structure to arrive at a model that better represents the data. Sub steps in such an approach are the modification of an existing structure (adding or deleting edges), the measurement of the quality of a given structure and an overall search algorithm guiding the whole learning effort. Strategies for each of those sub steps have been proposed and can be found in the literature.

A normal Bayesian network is static and is not able to treat the notion of time or sequences. In order to be able to model dynamic or time, dynamic Bayesian networks can be used. Dynamic Bayesian networks are introduced in [66] and Hidden Markov models and Kalman filters are as special cases of them.

Learning with hidden variables (variables that cannot be observed directly) can be done using an expectation maximization (EM) approach. In this two stepped approach, values of hidden variables are calculated based on the current parameter values (E step).
D4.1: State-of-the-art of event correlation and event processing

Then, the parameter values are recomputed based on the values of the hidden parameters calculated in the former E-step and maximizing the available data (M step). These two steps are carried out iteratively until some converging state is achieved. In case of Bayesian networks, the hidden variables are the values of unobserved variables for each example and in case of hidden Markov models, the hidden variables are the state transitions between the states.
4 Existing Systems and Applications

This chapter focuses on analyzing real implementations of event correlation frameworks and some applications using these frameworks as final systems applying event correlation.

4.1 Event Correlation Frameworks

Describing the state of the art of event correlation requires a complete research on the existing technologies available for this task. The research has been focused on general purpose systems, evaluating each of them paying special attention on the following criteria:

- CEP and/or ESP support
- Real-time pattern matching
- Open source / commercial implementations
- Implementation language
- Graphical User Interface (BRMS)
- Input event formats
- Event correlation technique
- SQL / rules style
- Capabilities and strengths

We are going to focus on general purpose event correlation platforms. Some correlation engines are built for specific domains, with the advantage of special operations and data structures for that domain (e.g. in the domain of network management: specific functions to evaluate whether two hosts belong to the same network [55]). Nevertheless, no engine has been developed with the purpose of processing software maintenance events. Hence, we have chosen multi-purpose platforms and we will take a look at some domain aware correlation engines in the section dedicated to existing systems applying event correlation.
4.1.1 Existing Open Source Event Correlation Software

4.1.1.1 Borealis Stream Processing

Developed at Brandeis University, Brown University and MIT, Borealis [1] is a second-generation distributed stream processing engine as an evolution of Aurora and Medusa, taking from the first the core stream functionality and the distribution functionality from the second. Both Aurora and Borealis are general-purpose data stream management systems, in order to support real-time monitoring applications. The approach of these systems is ESP, since it claims to be stream oriented, with data flowing through a network of nodes as tuples, processing the data at the boxes of these nodes. One of the main strengths of this system is the optimization of distributed processing, deploying a network of cooperating stream processing engines, which distributes query processing across multiple machines, maintaining integrity even when the network is dynamically changed.

Regarding the source code, Aurora and Borealis are available through the Aurora and Borealis Project Homepages. Aurora is also the origin of a commercial system (Streambase). All the related projects are programmed in C.

Borealis takes dynamic revision of query results, dynamic query modification and flexible and highly-scalable optimization as main requirements. Dynamic revision of query results are needed for correcting errors in previously reported data or because data may arrive late and miss its window, specially in stream sources, such as sensors, where data may be unpredictable. Dynamic query modification is desirable to change certain attributes of the query at runtime, since the system may want to obtain more precise results on a specific subset of the data. The highly-scalable requirement tries to give a response to sensor heavy and server heavy problems, the two main challenges of an optimization framework in order to consequently tune up some QoS metrics, like latency, throughput or sensor lifetime.

Regarding graphical tools, Borealis provides a graphical query editor, simplifying the composition of streaming queries with an XML-based query definition language. It also comes with a system visualizer to dynamically display the Borealis network topology.

A Borealis system comprises the following components:

- Distributed Catalog: It holds information about the overall system, including the description of the query diagram and the deployment information, which specifies the assignment of operators to processing nodes.

- Nodes: Location where the actual stream processing is performed. Each node runs a fragment of the query diagram and stores information about that fragment in a local

1www.cs.brown.edu/research/aurora and www.cs.brown.edu/research/borealis
D4.1: State-of-the-art of event correlation and event processing

The main components of the Borealis nodes are shown in the following figure[1]:

The Query Processor (QP) is the core component where actual query execution takes place, feeding input streams into it and pulling results through I/O Queues. The Local Optimizer communicates with the run-time components of QP to give performance improving directions. Among those run-time components, we should mention the Priority Scheduler, in charge of determining the order of box execution; the Box Processors, one for each different type of Box and the Load Shedder, which discards low-priority tuples when the node is overloaded. The Storage Manager is responsible for storage and retrieval of data of the local query diagram. The Local Catalog stores query diagram descriptions and metadata, and is accessible by all the components. Queries are written using a textual (XML-based) query definition language.

The distributed engine is achieved by the use of the other server component, Borealis Node, which communicates with their peers on other nodes to take collaborative actions. NH (Neighborhood) Optimizer uses local load and other NH Optimizers information to
improve local balance between nodes. The High Availability modules on different nodes monitor each other and, in case one fails, another node takes the lead in processing. Performance-related data is collected by the Local Monitor and the Global Catalog holds information about the complete query network and the location of all query fragments.

The Borealis project is no longer an active research project, the last release was distributed in the summer of 2008.

4.1.1.2 Simple Event Correlator

Simple Event Correlator (SEC) is an open source event correlation tool written in Perl, which is placed in the group of the rule-based approaches for processing events. Some of this tool's objectives are platform independence, lightweight execution and simplicity of configuration, as well as applicability for a wide variety of event correlation tasks.

One of the reasons why SEC is written in Perl is the independence from operating system platforms, since it runs on almost every operating system. SEC consumes few system resources, since it does not need much disk space and its configuration is stored in regular text files, containing one or more rules.

The objective of lightweight execution is widely achieved by SEC. It can be installed even on low memory and CPU performance workstations. Performing hundreds of event correlation operations with hundreds of contexts simultaneously active and stored in the SEC working memory, the program consumes less than 5MB of memory [55] on most architectures.

SEC produces producing output events by executing user-specified shell commands over line-based input events from a file stream. This tool uses a regular expression language for recognizing input events regardless of their format. SEC can take regular files, named pipes and standard input as input events, with the advantage that it can be employed as an event correlator for any application that is able to write its events to a file stream. Besides regular expressions, custom Perl functions can also be used to match the input lines, or to evaluate conditions. The range of actions cover the creation of log messages, writing events to a file or even executing an external program.

SEC is basically ESP-oriented, without specified support for CEP operations. Nevertheless, it is also capable of applying basic matching operations to text-based input files, looking for patterns specified by rules written in Perl.

The basic correlation operations that SEC [55] provides are:

- Single: Matching of input event and executing one or more actions immediately.
- SingleWithScript: Matching of input event and executing an action list in case an external script or program returns an specific exit value.
D4.1: State-of-the-art of event correlation and event processing

- SingleWithSuppress: After matching input event and executing an action list (single operation), the matching sleeps for t seconds, ignoring the following matching events during that time.

- Pair: Matching of input event, execute an action immediately, and ignore following matching events until some other input event arrives. When the event arrives, execute another action.

- PairWithWindow: Pair operation with a window of t seconds: Match input event and wait for t seconds for other input event to arrive. If that event is not observed within the window time, execute an action. In case the event arrives on time, execute another action.

- SingleWithThreshold: Matching input events are counted during t seconds and if a given threshold of this count is exceeded, execute an action and ignore all remaining events during the rest of the time window.

- SingleWith2Thresholds: Count matching input events during t1 seconds and if a given threshold is exceeded, execute an action. After that, start counting of matching events again and if their number per t2 seconds drops below the second threshold, execute another action.

- Suppress: Suppress matching input events in order to keep the events from being matches by later rules.

- Calendar: Execute a scheduled action list (at specific times).

Apart from matching conditions and action lists, SEC optionally provides the use of boolean expressions of contexts. SEC contexts represent the knowledge that SEC has learned during the event correlation process. The role of contexts is the activation and deactivation of rules dynamically at runtime. The following actions regarding the context management can be performed in SEC:

- create: Create a context, and optionally set some of its parameters.

- set: Set the parameters of a context.

- delete: Delete a context.

- add: Associate an event with a context.

- report: Given all events of a given context; supply them for external processing.
Hence, more complex event correlation schemes can be defined by combining several rules with appropriate action lists and context expressions.

A graphical user interface is not available in SEC, only a command line interface (CLI) is provided.

SEC works on a variety of OS platforms, like most of Linux distributions and some of Windows, and it has been applied to network fault and performance management, logfile monitoring, intrusion detection and fraud detection.

4.1.1.3 Esper

Esper[55] is an open source component under the GNU GPL license for building real-time ESP applications in Java (additionally, NEesper, written in C#, can be used with .NET). It also claims to support CEP, but it is basically ESP-oriented. It works as a real time engine in charge of triggering actions when event conditions occur among event streams in the system. Rather of storing the data and running queries, Esper allows applications to store queries and run the data through. This statements can be added and removed dynamically, while the engine is running.

Esper is general purpose, and it has been applied to several areas such as business process management and automation, finance, network and application monitoring and sensor network applications. The engine has been tested on Windows XP and on Redhat Linux.

The core language is SQL-like, called EQL, which is highly oriented toward support of modern technologies, and easy to extend. This language includes support for real-time pattern matching, since it has been enriched with temporal event correlation, as well as a way to control the lifecycle of each pattern \(^2\). EQL provides ESP basic operations like filtering, joins and aggregation.

Input events can be represented by Java Beans, POJOs (Plain Old Java Objects), Maps, XML objects or simple name value pairs. Esper tries to keep the minimum number of events in memory, because the engine can decide which events are required. The data that the engine retains is based on the queries registered with the engine, so it only selects the minimum needed events to satisfy any started statements. Event queries and pattern statements are registered in the Esper core container. Then events flow in at real-time speed and trigger arbitrary logic bound to the engine in the form of POJOs.

Currently, Esper doesn’t have a graphical user interface management as other tools provide, it neither provides query editor or portal application.

Some use cases of Esper are Holmes, Rocksteady and Esperr. Holmes [21] is an event-driven solution to monitor data centers through continuous queries and machine learning

\(^2\)http://esper.codehaus.org/
used, for example, to monitor the Internet site of a Brazilian TV reality show, called Big Brother. Every week this site receives users voting to eliminate a participant of the show. In the final week a new world record was broken with 151 millions of votes in two days. Holmes prevented an attack that could interrupt the real-time voting during the final week voting period. The authors of Holmes obtained some conclusions and future works, and they wrote about Esper: “As our future work, one challenge that needs to be addressed is the creation of interface to help IT operators create and test queries to Esper”, which remarks the mentioned lack of graphical user interface.

Rocksteady is a Google’s open source effort that uses Esper. According to its blogspot: “Rocksteady can be used in a number of different environments, but here on the AdMob operations team, we use it to determine the cause of events such as latency. We monitor requests per second (rps) and a slew of other metrics such as CPU and network traffic, then put them together in a prediction algorithm such as Holt Winters to predict a confidence band for the next arriving value”

Esperr is composed of Esper and R. R is a free software environment that provides sophisticated analysis tools. In Esperr, Esper produces new output events when the EQL statement conditions are satisfied and then R uses these new events to statistical computing and graphics.

4.1.1.4 Triton

Triton is a rules-based event correlation system written in Java that solves problems of centralization, analysis, processing and storing of generated events coming from a wide range of agents. It allows the configuration of agent subscriptions to a selection of the available correlators. The knowledge base consist of a set of rules written in DRL (Drools Rule Language), and associated models.

The Triton components are:

- Transceptor: Sends events information generated by the agents. Uses XML-RPC protocol communication, but shortly it will be extended to SOA.

- Triton-ESB: Gathers event information sent via XML-RPC in an Enterprise Service Bus, recovering and sending it to the associated correlators. This component is also responsible for checking if the information belong to an existing model. Otherwise, it would deflect the event to a default correlator, which correlates lost events.

- Triton-Service: Implements the correlation service of the system, using Drools 5.0.1 as supporting technology. Triton-Service is implemented over Spring Framework,
allowing the creation of several correlators. Each correlator is associated to a set of models, and each model has an associated JMS listener in charge of injecting facts in the working memory. Depending on the nature of the required analysis, correlators can be time-aware or not. The control of correlators is provided by the use of deployers, allowing them to be deployed from the Triton-Web interface.

- **Triton-Web**: Improved BRMS based on Guvnor open source. Allows the creation of correlators, its associated models, guided design of rules, DSL configuration connected to the models, deploying of correlators and the agent subscription to correlators. Among other features, we should mention Triton dashboard, a graphical interface with statistics of input and output events, correlators status, event queues of each correlator and monitored system status. It also allows the copy of resources from a correlator to other, improving the creation of correlators.
Triton enhances a business rules management system called Drools in order to give business users and analysts the ability to make routine changes and updates to critical business systems. The idea is that, using common business terms and familiar interfaces, business users and analysts can update business strategies across enterprise information systems. The final advantage of this control of the business logic embedded in IT systems is the capability of altering application behavior, which is provided to business users without IT assistance.

Drools is a business rule management system and an enhanced rules engine implementation (ReteOO), based on Charles Forgy's Rete algorithm tailored for the JVM[55]. It is based on forward chaining inference written in Java. Forward chaining is one of the two main methods of correlation when using inference rules (the other one is backward chaining). Forward chaining starts with the available data and uses inference rules to extract more data until a goal is reached.

Drools, as an inference system using forward chaining, searches over the inference rules until it finds one where the antecedent ('if' condition of the rule) is known to be true. After an antecedent is found to be true, it can infer that the consequent is also true, resulting in the addition of new information to the available data, iterating through this process until the goal is reached.

We must understand a couple of concepts before going deeper inside the Drools engine. The first of them is the concept of fact: it can be conceived as the container used with the purpose of transporting information into and out of the rule engine. Programmatically, the concept is better understood if we just talk about objects containing the events reaching the rule engine[14].

The other main concept that we must introduce is the working memory. In real life, enterprise applications will have more than one user (or process) calling rules at the same time. Then, using the same memory would be really confusing. Working memory is the solution to give each of them their own workspace in order to insert, update or retract facts. In other words, rules change facts in the working memory.

The use of the mentioned enhanced implementation of the Rete algorithm, which replaces all the if-then statements with an optimized network. Rete allows faster execution of rules by sorting them in such a way that when facts change, the rule engine is immediately prepared, knowing which rules have to be fired[14]. Between the advantages that Rete contributes to Drools, we can mention the reduction or elimination of certain types of redundancy, the storing of partial matches when performing joins between different fact types and the efficient removal of memory elements when facts are retracted from working memory.
Triton supports CEP and partially ESP approaches. It provides real-time pattern matching, since it has already been applied as Intrusion Detection System, as well as for network management purposes.

Currently, Triton development is focusing on an in progress open source distribution, making it easy to integrate with multiple systems.

### 4.1.2 Commercial Event Correlation Software

#### 4.1.2.1 RuleCore

RuleCore\(^5\) is an event-driven reactive rule engine written in Python which provides a complete set of GUI tools\(^{[55]}\) for rule building and composite event definitions.

It was designed for Complex Event Processing, focusing on real-time rule-based detection of business situations (complex event patterns), which must be detected and reported by automatically evaluating user defined rules in response to inbound business events. In response to incoming events, ruleCore's engine is evaluating reaction rules continuously. These rules are defined using the Reakt language, which is a language designed for expressing business situations. RuleCore claims to be "no programming required", since the Reakt language is a high level declarative XML based language. These rules can be created, removed, modified and monitored in a ruleCore server. The event model is also specified using XML schema descriptions and XML is used as the event representation.

Inbound events come from one of the large number of pluggable event transport protocols to the system through the Event I/O module and they are transformed as outbound events. Each outbound event is the notification provided to external systems about a detected situation. The last event in the sequence of inbound events has a special meaning, because the situation is said to be caused by this event and the ruleCore CEP Server adds meta-data to keep this information. The whole procedure can be started again, since outbound events can be inbound events for other rules, so inbound events can also

\(^5\)http://www.rulecore.com/
D4.1: State-of-the-art of event correlation and event processing

carry metadata indicating what event caused them. Thus, in a certain way, it's possible to determine the root cause of an event, by following the event causality list. Some situations can consist of a long sequence of events. Sometimes, before the situation is fully detected, these events must be tracked for long periods. The ruleCore CEP server provides automatic recovery procedures to ensure the integrity of long-lived instances and restore all events and rule instances and their situation detectors in case of sudden server reboots or crashes.

Another key feature is the situational monitoring. The state of the ruleCore CEP Server (rules, situations and actions) can be monitored using events as monitoring queries. The ruleCore CEP server can process data from GPS, AIS or other sensors, because ruleCore language defines event types for sending location data into using the WGS84 standard, which is used, for example, by GPS⁶.

4.1.2.2 ILOG

WebSphere ILOG JRules⁷ is a general-purpose rule engine that combines rule-based techniques and object-oriented programming to help adding rule-based modules to business applications. The ILOG JRules engine provides efficient ways to add decision support and data flow control functions to business applications. It business focused tool for developing intelligent agents and business rule processors.

While traditional rule-based systems often require a proprietary language to define the objects used by the inference engine, the ILOG JRules engine directly infers from the C++ objects of a business application without any duplication. The architecture of the application using ILOG can be dissociated from its rules altogether, so it can implement rule-based components for carrying out high-level decision making and control tasks.

The ILOG JRules engine also uses the previously mentioned Rete algorithm, providing efficient performance of pattern matching on different conditions. The ILOG JRules source code is implemented in C++ and works in a completely object-oriented way.

ILOG JRules offers a web-based Business Rule Management System, with several features like the Rule Team Server for collaborative business rule repository and the Rule Execution Server, a J2EE environment to deploy business rule applications and SOA-based decision services. It also provides a testing environment to create and store business scenarios applying business rules to each of them.

⁶http://rulecore.com/content/view/15/34/
D4.1: State-of-the-art of event correlation and event processing

Historically, rule-based systems were used to develop knowledge-based and expert systems. The present proliferation of computer networks as global business and communication networks, for example the World Wide Web, company intranets and business to business networks, provide environments that requires various and numerous expertise, which can be implemented with rule-based modules. This is the main target of this system, allowing easy integration with business applications.

The ILOG JRules engine provides support for developing both business rules applications and telecommunications applications. For example, an application using business rules to personalize client demands can benefit from a rule-based system where various client characteristics and requirements could be used to filter information from a database. For telecommunication applications, any network surveillance program can take advantage of the temporal correlation, transition monitoring and alarm handling facilities available using the ILOG JRules engine. It successfully synthesizes the paradigms of rule-based and object-oriented programming to produce intelligent applications.

WebSphere ILOG BRMS stores rules in a central repository. The same rules can be accessed across the enterprise, across touch points and applications. Teams employ consistent regulations and practices, ensuring that systems follow compliance requirements.

Input events are formatted at the CEI[84] (Common Event Infrastructure) Event Source, which is a component implementation for unified formatted events, no matter if they are business, system and network events. The unified format is the Common Base Event format. This component allows sending information to the CEI Server from where applications can subscribe to particular types of event.
Among several features, ILOG allows to express business rules with a close-to-natural language, so business analysts can manage, track and change rules themselves.

**Oracle Complex Event Processing**

Oracle Complex Event Processing\(^8\) is a component from Oracle SOA Suite (a key member of the Oracle Fusion Middleware family of products) which was voted as the number one Complex Event Processing Solution Provider by Waters Ranking\(^9\). It is a lightweight Java application container based on Equinox OSGi.

As a service engine, Oracle CEP allows user to aggregate, correlate, enrich and detect patterns through the Continuous Query Language (CQL)\(^{[25]}\) to define streams, relations, views, register event sources and destinations. The pattern matching feature is also ensured by this language, providing an Oracle CQL MATCH_RECOGNIZE condition with several clauses to express complex conditions among stream elements.

Investigating possible input event formats for this system, we found out that Oracle CEP claims to be “Hot-pluggable”, in other words, it can analyze events across heterogeneous system sources. Besides, Oracle CEP can recognize trends across a huge number of events within a given time frame, it can detect missing events, or events that should have occurred but did not.

There is an advanced run-time administration in the suite, called Oracle CEP Visualizer, which has been created to manage, tune and monitor applications, displaying information about management events and performance monitoring.

---


D4.1: State-of-the-art of event correlation and event processing

Oracle CEP works by listening to event streams, processing them and generating notable events that are received by event sinks. These event streams are evaluated through CQL queries, performing filtering and aggregation functions to discover and extract the output notable events.

The Oracle CEP application is composed by the following elements:

- **Adapters:** Which understand the inbound and outbound protocol, so they convert the event data into a normalized form that can be queried by the processor.

- **Channels:** Event processing endpoints, responsible for queuing event data until the event processing component can act upon it.

- **Processors:** Which consume normalized event data and process it using CQL queries, generating event to an output channel.

- **Beans:** In charge of listening to the output channel to forward the generated events to external event sinks.

As we explained previously and to sum up, some of the main benefits of this system are real-time pattern matching support, heterogeneous event sources and a high-scalability and availability, which makes of Oracle CEP one of the most powerful tools in the market.
4.1.3 Summary

With the purpose to sum up the research of existing event correlation systems, we provide in the following tables an overview of the features of the discussed systems. Rather than trying to list all capabilities, we have pointed out the most noteworthy features, based on the following criteria list:

- CEP and/or ESP support
- Real-time pattern matching
- Open source / commercial implementations
- Implementation language
- Graphical User Interface (BRMS)
- Input event formats
- Event correlation technique
- SQL / rules style
- Capabilities and strengths
### Table 4.1: Comparison of existing event correlation frameworks (Part 1 of 2)

<table>
<thead>
<tr>
<th>CEP/ESP</th>
<th>GUI</th>
<th>Pattern matching</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP</td>
<td>Query editor and system visualizer</td>
<td>Supported</td>
<td>C (GPL)</td>
</tr>
<tr>
<td>SEC</td>
<td>Not available (command-line interface)</td>
<td>Only text-based</td>
<td>Perl</td>
</tr>
<tr>
<td>Esper</td>
<td>Not available</td>
<td>Supported</td>
<td>Java and C#</td>
</tr>
<tr>
<td>Triton</td>
<td>Complete Web UI (Guided rules, DSL editor, Model definition)</td>
<td>Supported</td>
<td>Java</td>
</tr>
<tr>
<td>RuleCore</td>
<td>GUI for rule building and composite event definitions</td>
<td>Supported</td>
<td>Python</td>
</tr>
<tr>
<td>Borealis</td>
<td>Supported</td>
<td>Commercial (closed)</td>
<td>C++</td>
</tr>
<tr>
<td>Esper</td>
<td>Supported</td>
<td>Commercial (closed)</td>
<td>Java</td>
</tr>
<tr>
<td>Triton</td>
<td>Supported</td>
<td>Commercial (closed)</td>
<td>Oracle CEP Visualizer</td>
</tr>
<tr>
<td>Oracle CEP</td>
<td>Supported</td>
<td>(MATCH-RECOGNIZE condition provided by Oracle CQL)</td>
<td>Java</td>
</tr>
</tbody>
</table>

**D4.1: State-of-the-art of event correlation and event processing**
### D4.1: State-of-the-art of event correlation and event processing

<table>
<thead>
<tr>
<th>Input event format</th>
<th>Event correlation technique</th>
<th>Style (SQL-based/rule-based)</th>
<th>Capabilities and strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borealis</td>
<td>Network of processing nodes</td>
<td>SQL (queries defined with XML-like language)</td>
<td>Distributed processing, dynamic revision of query results, dynamic query modification and flexible and highly-scalable optimization.</td>
</tr>
<tr>
<td>ESPer</td>
<td>Rule-based correlation</td>
<td>SQL-based (DRL)</td>
<td>High-speed execution of SQL-like statements.</td>
</tr>
<tr>
<td>Triton</td>
<td>Rule-based correlation</td>
<td>Rules-based (JReakt XML)</td>
<td>Efficient algorithm (Rete-based), allows the creation of DSL, integrated BRMS, Automatic discovery of agents.</td>
</tr>
<tr>
<td>RuleCore</td>
<td>Rule-based correlation</td>
<td>Rules-based (Java)</td>
<td>Situation monitoring, no programming (only XML).</td>
</tr>
<tr>
<td>ILOG</td>
<td>Rule-based correlation</td>
<td>SQL-based (CQL)</td>
<td>DSL support, Rete algorithm, decision logic represented in text and graphics.</td>
</tr>
<tr>
<td>Oracle CEP</td>
<td>Rule-based correlation</td>
<td>SQL-based (Hot-pluggable)</td>
<td>Real-time pattern matching support, heterogeneous event sources, and a high-scalability and availability.</td>
</tr>
</tbody>
</table>

**Table 4.2: Comparison of existing event correlation frameworks (Part 2 of 2)**
4.2 Event Correlation Applications

In this section, we will provide an overview of systems that have been created or updated to give response to the needs of event correlation in a specific business area. These system descriptions can be useful to provide the way to raise the event correlation task in several areas like network and security management, which can be useful since our area of application (software maintenance) has some items in common with the applied business areas.

4.2.1 OSSEC

For starters, we are going to focus on one of the systems targeting security management, specifically intrusion detection. OSSEC\textsuperscript{10} is an open source host-based IDS (Intrusion Detection System). The main features it provides are file integrity checking, rootkit detection, log monitoring and active response. About the outputs OSSEC can give, we should mention storing events in a database, sending email, generating reports, logging to syslog, Windows registry monitoring, real-time alerting and access to results to a web user interface.

Narrowing the analysis to the way it performs correlation, it is implemented in the analysis daemon (analysisd). The approach of the correlation is rule-based, and it uses XML rules (also rules written in C), each of them with a unique id, a level of priority of the match and conditions for matching. The action performed through the rule depends on the level (from 0 to 15). On level 0, no alerting is done, whereas on higher levels the message is printed on syslog or even sent to an administrator\textsuperscript{53}. The XML implementation allows the creation of a tree of rules, where some of them are conditional to others, since they can only match if those have matched. The reason for such nesting is efficiency, in a similar way as the Rete algorithm, although the rule tree is created manually and it only prevents some rules to be evaluated if it is not needed (only the first rule of the tree is evaluated if it does not match).

Although OSSEC is a great IDS tool, the correlation operations possible with XML rules are rather basic. Nevertheless, more complex rules can be developed using C language, but the creation and maintenance of a rule in that case is more difficult and time-consuming, with one more disadvantage, a recompilation is required for each change in the rules.

\textsuperscript{10}http://www.ossec.net
4.2.2 HP Event Correlation Services

HP OpenView Event Correlation Services (ECS) is an event correlation technology designed to deal with event storms in the telecommunications environment. ECS is integrated with Network Node Manager (NNM) to correlate network related events and HP OpenView Operations (OVO) to correlate events originated in the different layers of the network equipment: systems, applications, databases, and the Internet. This layered approach provides a distributed management solution, resulting in reduced network bandwidth usage and cost savings.

The Event Correlation Services are composed by:

- ECS Engine (run-time)
- ECS Designer (developer)
- ECS Protocol Modules (CMIP, SNMP, ASCII)

These components can be properly customized, and they perform specific correlation functions. Complex correlation circuits can be built combining these components. These circuits represent the event flow in a flow graph, as a network of processing nodes, represented together with any associated data and relationship information. Nodes can be used for simple operations, such as filtering the event stream; more complex nodes can be created as a combination of multiple primitive nodes.

The ECS Engine is a run-time correlation engine. ECS executes a set of correlation rules which control the processing of event streams. The ECS Designer is the graphical user interface that is used to develop correlation rules. Selecting, connecting and configuring logical processing blocks are some of the features offered by this GUI to develop rules interactively. Correlations rules can be tested in the ECS Designer too. The ECS Protocol Modules let ECS Designer and the run-time ECS Engine define and support the event protocol (CMIP, SNMP, or ASCII).

There are two kinds of nodes: primitive and compound. Primitive nodes have a predefined unique logical function (there are fifteen primitive node types). Compound nodes encapsulate a subset of a correlation circuit which defines a new node type with user-defined functionality. Each node has a set of ports, which are used to connect this node to another node. There are lot of types of ports: input, output, error, reset, and so on. For example, some events enter a circuit and flow through, if a run-time error occurs in evaluating the expressions configured for particular node instance, an event will be output through an error output port. The same way, there are primitive and composite events. Primitive events is a single event that could be created within the engine or entered.

\[11\]http://findarticles.com/p/articles/mi_m0HPJ/is_n5_v47/ai_18895247/
\[12\]http://h20229.www2.hp.com/products/ecs/
D4.1: State-of-the-art of event correlation and event processing

into the correlation engine. ECS supports (Common Management Information Protocol, ISO/IEC 9596-1) and SNMPv1 (Simple Network Management Protocol, version 1) event types. Composite events allow multiple events to be collected into a single addressable structure in which all members are accessible. This kind of events can be only defined within a correlation engine, and can be passed into an out of compound nodes, but cannot be output from a top-level circuit back environment. There are a third event type: Temporary events, which are used when an event is required as an internal container for data, or when a trigger event is required. The scope of temporary events is the same as the composite events.

The arriving events, their information and the current time of the engine are used by the nodes of a correlation circuit to make decisions, but there are more data available that the node also uses: node attributes, the data and fact stores and annotation data

Node attributes: Depending of its type, a node can export different attributes to other nodes throughout the circuit. For a better understanding, this quote from an HP Journal’s article: “The count node exports a count attribute which increments or decrements for each arriving event. The table node exports two attributes: a count attribute whose value is the number of events currently stored and a contents attribute whose value is a list of all events stored in the table.” If the node is compound, it can export the attributes of any contained nodes as attributes of the compound node.

Data and Fact Store: They contain entries which are accessible globally throughout the correlation circuit. The data store entries are referenced by name, so their values need not be known when the circuit is designed. This feature allows correlation circuits to be reused at multiple sites. The fact store allows that node parameter conditions and expressions test the relationships between objects.

Annotation: Data from outside can be obtained by an annotate node for use within the engine. In addition to make correlation decisions, this information can also be added to created or modified events.

A fundamental feature of ECS is that event information can be enhanced. It means that all available information is consolidated from multiple events, external data, data store and fact store, and all superfluous data is discarded. So, when all pertinent information has been assembled, it must be forwarded creating a new primitive event or modifying the data values in an existing primitive event, using create or modify node, respectively. Those input multiple events, each containing only a fraction of the total relevant information, can be suppressed. The result is that the events actually delivered to management systems contain enhanced information content.

ECDL is a complex and sophisticated language which supports the complete specification of the correlation circuit (including dynamic node expressions and conditions). ECS

13http://findarticles.com/p/articles/mi_m0HPJ/is_n5_v47/ai_18895247/
Designer produces ECDL code which is encrypted. Direct coding using ECDL is not supported and cannot be compiled.

Even though HP ECS can be seen as a rule-based system, it does not rely on ECA rules, but instead uses rules to control the event flow.

### 4.2.3 OpenNMS

OpenNMS is an open source network monitoring platform written in Java. According to its website, the main focuses of OpenNMS are service assurance, event and notification management and performance measurement.

OpenNMS is clearly oriented to Network Management, including a discovery process of any element within the network topology. In that sense, the discovery process can also generate a new Suspect event when an interface responds to a ping.

The current version is 1.8.4, and the strategy of the correlation is Rules Based correlation, using the Drools engine. Accessing the open source SVN we find the code related to the Drools correlation, and we notice that it deals with flap control, root cause analysis or possible causes.

The system manages events through a process (daemon) called eventd. It deals with two main types of events: internal and external. The internal events are generated by the OpenNMS software and the external events are generated via external SNMP traps. The daemon listens on port 5817, which allows even these external processes to send events to the system.

In the event generation, parameters like description, severity and log message can be set and, in addition to that, automatic actions can be launched to send event parameters to an external script. These parameters can be configured through the eventconf.xml file. This file contains every event definition in the system, as well as the universal event identifier (UEI), the description, the log message to be provided once the event happens and the severity on each type of event.

Since version 1.3.3, OpenNMS is enhanced with rule based correlation and it allows integration with Drools, with the main purpose of allowing distributed monitoring. In addition to that, it added some new correlators, like the Isolated Service Flapping Detection, which solves the problem of the detection of abrupt variations in measures that should not be taken into account (flapping). For the sake of achieving the integration with Drools and other external correlation systems, OpenNMS specifies the engine to be applied with the correlation-engine.xml configuration file. In case Drools is selected, a configuration is produced with another xml file: drools-engine.xml, which contains the

---

14[http://www.opennms.org](http://www.opennms.org)

rule set configuration and whose events get injected into working memory for each rule set.

De-duplication of events can be done with the use of the reductionKey tag, identifying the event as an Alarm, since Alarms duplication in OpenNMS is prevented in order to decrease search times. It gives the ability to indicate which events are important and they become alarms, reducing these events to one row in the alarms table, and including a column with the count of the occurred events. As a benefit, the user can view only those events that actually represent problems and he immediately sees how many of each have been received and almost all of the events in one view.

Another OpenNMS feature is event substitution, usually employed to convert SNMP traps into events. Do not confuse substitution with event translation, other towards transforming and enhance the data contained within the OpenNMS events. These transformations can come from the result of SQL queries, regular expression matches and string literal that are then assigned to attributes of the new event. Like other OpenNMS daemons, the EventTranslator has a configuration factory class that marshals its configuration from an XML formatted file, called translator-configuration.xml. This file contains elements and attributes that create or change the fields of one event and publishes a new event with these changes.

OpenNMS also comes with a complete Notification System, where particular events can be selected to send a notification. Notifications can be done via email, Short Message Service (SMS), Extensible Messaging and Presence Protocol (XMPP), or any external program. Additionally, the data can be viewed in the web based Graphical User Interface (GUI).

Automation in OpenNMS is provided (since version 1.3) allowing, as an example, to increase the severity of some alarm if it has not been acknowledged in a period of time. A daemon called vacuumd runs SQL statements on an interval, allowing configuration of processes (called Automations) that are defined using Triggers and Actions. An automation is made up of trigger and action statements. Triggers are SQL statements whose results are analzyed and processed by the action statement, which is always required for an automation and it can be executed against the result set of a trigger or it can be executed independently. Automation’s configuration is performed over the vacuumd-configuration.xml file.

It provides an existing XML-RPC implementation with 6 specific events (associated to network management), for forwarding events to a server,. The sample configuration allows exporting these events via six specific XML-RPC method calls to a single server. In case that more that one server is listed in the xmlrpced-configuration.xml file, it would treat them as serially redundant, considering the first as the primary server and the others as backups.
Event data collection can be done via various protocols [55], such as HTTP, Simple Network Management Protocol (SNMP) or Java Management Extensions (JMX), and event sources include the following:

1. SNMP traps
2. Syslog messages
3. NESSUS (inputs from a vulnerability scanning program)
4. SNORT (lightweight network intrusion detection system)
5. Java Management Extensions
6. E-mails
7. Windows Event Log (configured to send SNMP traps)
8. Internal events

4.2.4 Open Source Security Information Management

Returning to the subject of the event correlation systems specializing in security management, OSSIM\(^1\) is an open source Security Information Management software under BSD (Berkeley Software Distribution) license, designed to offer a security management suite with integration of various open source software components (such as Nmap, Nessus, Snort, Nagios, OSSEC and others).

About OSSIM's core, responsible for event collection, management and correlation, as well as for risk assessment and alerting, it is written in C. As Andreas Müller claims in his research, OSSIM provides three different types of correlation:

- Cross Correlation. It relies on information about vulnerabilities on the destination host, in order to adjust the probability of a successful attack (reliability).

- Inventory Correlation. It relies on generic host information, such as OS, port, application, protocol to recalculate the reliability of an attack.

- Logical Correlation: It relies on memory about previously matched directive events. For this type, OSSIM makes use of XML directive, which are similar to the explained nested rules in OSSEC. In this case, each new event is matched against all directives, generating multiple alarms. These directives can create new events, which are the mentioned directive events. Hence, this correlation is rule-based.

\(^1\)http://www.alienvault.com/community.php?section=Home
4.2.5 Prelude–Intrusion Detection System

PreludeIDS is an universal SIM (Security Information Management) application written in C\[55\], focused in Intrusion Detection, collecting, normalizing, correlating and reporting all security-related events.

Security events are normalized to a single format: “Intrusion Detection Message Exchange Format” (IDMEF)\[17\], an international standard to enable interacting with the various security tools available on the market.

Prelude is divided in components\[18\]. Intrusion detection is provided by sensors, which report events to the Prelude-manager server in a centralized architecture.

Prelude-manager processes these events providing log analysis, correlation, event management \[55\], delivering some of them to a database or XML file. Finally, the Prelude console written in Python where the results can be visualized, and reports or alerts can be generated in various formats.

Prelude claims to be agentless, so rather to use agents it gathers input data from other security tools. Native compatibility is provided for a number of programs (such as OSSEC, Snort, PAM, Nepenthes) and anybody can write their own sensors or use some of the 3rd party available sensors. Prelude filters, classifies and correlates the data.

Correlation in Prelude is rule based, with correlation rules written in Lua, a lightweight multi-paradigm programming language, providing a small set of general features that can be extended to fit different problem types. In the prelude-manager, the component in charge of correlation, Prelude-Correlator, is a Python rules based correlation engine distributed under a GPL license agreement.

The engine allows the quick identification of main security events, correlation of alerts originated from heterogeneous sensors and real-time analysis. This engine is distributed with a default set of correlation rules\[19\].

\[17\]https://dev.prelude-technologies.com/
\[18\]https://dev.prelude-technologies.com/wiki/prelude/PreludeArchitecture
4.3 Related EU projects

For the sake of accomplishing the event correlation outlook, we must pay special attention to ongoing projects. One of the main meeting points of this kind of projects is the European Union Framework Programmes for research and technological development, currently the Seventh edition (FP7), which main objective is to gain leadership in key scientific and technology areas, and which is also the framework where FastFix, our current research project, is hosted.

One of the main projects related with advancing in the evolution of event correlation is ONTORULE. In the FastFix environment, one of the elements that can benefit the acquisition of knowledge are ontologies, specially if they can be properly combined with correlation rules, as we have seen in section 3.2 as a related concept of the event correlation research. In such terms, ONTORULE project can give an approach of the feasibility of a combination between ontologies and rules.

4.3.1 ONTORULE

ONTORULE\textsuperscript{20} is based on the slogan “ONTOlogies meet business RULEs”, centering on the research of acquisition of ontologies and rules, from several sources including natural language and documents, enabling the use of inference engines that combine ontology and rule-based correlation.

Trying to give response to this work, ONTORULE plans to create an integrating framework, based on the SBVR standard\textsuperscript{21} (Semantics of Business Vocabulary and Business Rules) to enable modelling and acquisition tools for ontology vocabularies and business rules. SBVR is an adopted standard of the Object Management Group, to allow business people to define the rules related with their business in their own language. This standard would help to capture these rules in a way that is clear and translatable into other representations. ONTORULE would use the SBVR standard to create an assisted framework which can guide the user in the creation and acquisition of business rules and the update of ontologies.

One of the main ideas that ONTORULE provides is the clean separation of the domain ontology from the actual business rules, hence the representation of the knowledge from its implementation. Nevertheless, it tries to go beyond this separation, since it also claims that the business rules, and the underlying ontology, must be acquired from natural language sources, this implies that the relevant people in the organisation must be able to manage and maintain onto- logies, business rules and data models separately.

As we introduced in Section 3.2, one of the main event correlation techniques shown

\textsuperscript{20}http://ontorule-project.eu/
\textsuperscript{21}http://www.omg.org/spec/SBVR/1.0/
D4.1: State-of-the-art of event correlation and event processing

```
rule Rule5 {
  when {
    this thing isa audi:Function
    | audi:Method
    | audi:MethodInputOutput;
  }
  then {
    print("Rule5:: This is an audi thing:");
    println(thing);
  }
}
```

Figure 4.8: PR(OWL) Rule Example

In the present document, the rule-based correlation, is characterized by the use of non-standard rule formats that are not compatible with semantic web standards. One of the high level objectives of ONTORULE, and probably the most remarkable in event correlation, is the research on efficient execution of rules and ontologies in inference engines, able to combine them.

Eiter[20] performed in 2006 a previous research about this combination of rules and ontologies, reaching an interesting analysis of some approaches combining rules and ontologies, concluding that a lot of work should be done for an efficient implementation.

At present, ONTORULE, gives the current overview of possible efficient combinations between rules and ontologies, concluding in three approaches: based on DL programs, F-Logic and F-Hybrid knowledge bases. It also concludes that Rete-based production rules systems do not activate rules based on semantic and logical entailment[32], they are based on clever pattern matching of explicit objects populating a working memory. The benefit of those systems is the efficiency of the procedure for deciding what objects are matched in the working memory thanks to the Rete algorithm.

ONTORULE studies also performed an overview on condition/action rules over ontologies, which are just prototypes at this moment. One of the approaches is PR-OWL[10], an initiative taken by IBM for using traditional rule-based engines of the IBM Production Rules Suite (JRules), explained in the section 4.1.2.2 of the present document, using PR-OWL, a prototype PR language developed by IBM specifically for ONTORULE.

PR-OWL specifies PR applications over OWL ontologies[32], instantiating a generic production rule language over OWL using the OWL-API. The rule definition is almost the same as the Drools syntax, and it is shown in an example of the Audi project in figure 4.3.1. As a rule definition language is allows to create statements and patterns, but also deal with ontologies with data-specific patterns, values, descriptions and actions in OWL. These OWL related objects will deal with the OWL reasoner, and the results will be injected to the working memory.
Heymans[32] identifies some of PR(OWL) issues and limitations, one of which is associated with discrepancies between OWL and objects. Production rules languages work over objects, assuming a closed world, while OWL ontologies assume an open world. In formal logic, the open world assumption refers to the idea that the truth-value of a statement is independent of whether or not it is known by any single observer or agent to be true. On the other hand, the closed world assumption holds that any statement that is not known to be true is false. Hence, in OWL a fact is considered as false only in case it is asserted to be explicitly.

Another discrepancy between OWL and production rule objects is that, in traditional rule-based correlation engines, variables in the condition area of a rule may match with collections (not only with single elements), while in the logical framework (OWL), it is only possible to match variables with single elements, so actions are performed based on tuples.

One of the notable results of PR(OWL) is that dealing with rule conditions, the knowledge implicit in the OWL knowledge base is not available immediately, but the PR system must probe every time the OWL reasoner for known facts by sending explicit queries. This happens in such a way that only the relevant facts to answering the query will the ones that will materialize in the working memory as a result. Thus, this prototype only has partial information of the whole knowledge base and having the whole sets of knowledge available in the working memory would ruin the effectiveness provided by the Rete algorithm.

4.3.2 Other EU Projects Related with Event Correlation

Finally, in this section recovers other EU projects applying event correlation which are currently being developed and whose starting date is too recent to have concluding results. Nevertheless, it is interesting to keep one eye on them, since they are the present and future of event correlation systems.

4.3.2.1 MASSIF

Located in the area of Security Information and Event Management (SIEM), MASSIF\textsuperscript{22} is an EU project in the scope of the 7th Framework Programme with the objective of providing a SIEM framework supporting intelligent, scalable and multi-level domain security event processing and predictive security monitoring. It will try to enable the detection of upcoming security threats and trigger resolution actions, even before the security problems happen. Hence, it will employ event correlation techniques for high performance event collection and processing, in the context of attack models.

\textsuperscript{22}http://www.massif-project.eu
4.3.2.2 SCAMSTOP

SCAMSTOP\textsuperscript{23} is an environment for automatic fraud detection in voice over IP networks, with the purpose of giving assistance to providers in detecting anomalous behaviour. It shall provide monitoring tools to automatically detect frauds and generate alarms to providers.

Two main lines will be developed as strategy pillars. On the first side, SCAMSTOP will use behavioural modelling based on statistics and anomaly detection methods, with an already proven efficiency in the area of banking. On the other side, and totally related to our area of research, it will use innovative approaches of multi-protocol event correlation, keeping in mind the specific nature of VoIP protocols and components. The aim of the event correlation side will be a high detection rate as well as an optimized resource efficiency.

\textsuperscript{23}http://www.sme-scamstop.eu/
5 Conclusion

The present state of the art document describes the most important techniques, frameworks, and applications on event correlation, as well as the main issues and ongoing research, focusing on the software maintenance domain. This section analyzes the most important findings in this area of knowledge, research and practice.

We found that event correlation has frequently been used in other areas than software maintenance, like network management or intrusion detection systems.

The most common and widely used existing technique is rule-based correlation, as it has transparent behaviour, and it allows to specify behaviour in a closer way to the natural language. In addition to that, control and knowledge is separated, so knowledge can be updated without changing the program code of the engine. Other techniques, like model based correlation requires a great knowledge of the domain, and the description of behaviour may be difficult. Bayesian networks are supported by a good theoretical foundation, although it requires coming up with prior probabilities before computation.

Ongoing research on semantic event correlation is quite promising, since it proves to correlate a percentage of events that could not be performed with traditional syntactic correlation. It also provides a way to model knowledge with ontologies, and use it in combination with rules, which is currently being investigated by the ONTORULE project. Nevertheless, their good results are currently a prototype and some limitations have been found in this combination, since the rule engine’s world and the ontology reasoner’s world have different nature, so incoming issues arising in its integration are yet to be solved.

One of the main issues detected in the event correlation discipline is the lack of automation to adapt to un-coded situations. In that sense, learning techniques can be useful in order to find patterns and shadowed relationships between events, as well as to discover new rules or maintaining the existing ones.

Finally, in the comparison of event correlation frameworks, where the research has been guided on a set of criteria to evaluate each of them, the results provide several systems that can support Complex Event Processing and Event Stream Processing in different levels, most of them rule based frameworks supporting real-time pattern matching. Only some of the solutions are flexible enough to gather multiple input event formats, customizing the model definition and applying efficient event correlation.
Bibliography


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


D4.1: State-of-the-art of event correlation and event processing


